DEPARTAMENTO DE TEORÍA DE LA SEÑAL Y LAS COMUNICACIONES

UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

SEGMENTACIÓN DE CAMPO DE JUEGO EN VÍDEOS DE BALONCESTO PARA DETECCIÓN DE EVENTOS EN TIEMPO REAL

AUTOR: JESÚS FERNÁNDEZ BES

TUTOR: JESÚS CID-SUEIRO

Septiembre 2010

Any sufficiently advanced technology is indistinguishable from magic.

(Cualquier tecnología suficientemente avanzada es indistinguible de la magia).

Arthur C. Clarke

Resumen

Las retransmisiones de vídeos deportivos, debido al gran atractivo que despiertan en una numerosa audiencia y a los importantes intereses económicos que existen en torno a ellas, llevan años siendo un importante motor del desarrollo y promoción de tecnologías tanto de comunicación como de procesado de imágenes. En este contexto existe un creciente interés por ser capaz de extraer características de secuencias de vídeos deportivos que permitan la detección de eventos del juego durante su retransmisión (en tiempo real) para enriquecer la información que se le proporciona al telespectador.

Este proyecto consiste en el estudio, diseño e implementación de una parte de un sistema de este tipo: un segmentador del campo de baloncesto. Nuestro objetivo es separar el campo de juego en secuencias de vídeo de baloncesto FIBA de manera robusta, extensible a cualquier campo de baloncesto, y eficiente, para poder integrarla en un sistema de detección de eventos.

En primer lugar se estudió el estado del arte en segmentación de campo, no sólo en vídeos de baloncesto sino también en otros deportes. A partir de este estudio se diseñaron e implementaron diversos prototipos evolutivos del sistema. De todos ellos se analizó su funcionalidad con sendos bancos de pruebas extraídos de vídeos de varios partidos. Finalmente se realizaron pruebas de rendimiento que nos permitieran evaluar la versión final en términos de robustez y eficiencia.

Aún sin cumplir plenamente los objetivos propuestos, los resultados obtenidos llevan al optimismo y nos permiten abrir nuevas líneas de investigación en este campo.

Abstract

Sport video broadcasts, as they appeal to large audiences and due to the number of economic assets involved around them, have been an important development and promotion driving force of both communication and image processing technologies. In this context there is a growing interest in being able to extract features of sport video sequences that allow the detection of game events during the broadcast (in real-time) to enhance to information delivered to the viewer.

This project consists in the study, design and implementation of a part of this type of system: a basketball court segmentation system. Our objective is to segment the court in FIBA basketball video sequences in a robust, it must be extended to any basketball court, and efficient way, so we can integrate it in a event detection system.

Firstly the state of the art of court segmentation was studied, not only of basketball videos but also of other sports. Based on this study some evolutive prototypes of the system were design and implemented. The Functionality of all of them was studied with some test benches extracted from a number of game videos. Finally, some performance tests were performed in order to evaluate the last prototype in terms of robustness and efficiency.

Although the stated objectives were not completely achieved, the obtained results give us reasons to be optimistic and allow us to open new research topics on this area.

Índice general

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

1.1. Presentación y motivación del problema

Las retransmisiones de vídeos deportivos, debido al gran atractivo que despiertan en una numerosa audiencia y a los importantes intereses económicos que existen en torno a ellas, llevan años siendo un importante motor del desarrollo y promoción de tecnologías tanto de comunicación como de procesado de imágenes. Algunos ejemplos de esto son la implantación del estándar 3G de comunicaciones móviles en Japón, la implantación de la HDTV (*High-Definition TeleVision*) o el desarrollo de diversos servicios web de vídeo bajo demanda [10].

En este contexto existe un creciente interés por ser capaz de extraer características de secuencias de vídeos deportivos que permitan clasificar, indexar o etiquetar automáticamente dichas secuencias para ser capaces de llevar a cabo una navegación y unas búsquedas más inteligentes sobre la creciente cantidad de las mismas [21].

En algunas de estas aplicaciones, como son las que tienen como objetivo la detección de eventos del juego durante la retransmisión, se necesita además que ésta extracción de características se realice en tiempo real (o casi tiempo real). En este contexto denominamos eventos semánticos o simplemente eventos a aquellas escenas del vídeo que resultan de especial interés para el espectador. Dado el interés de estas secuencias, resulta útil detectarlas durante la retransmisión para que así puedan ser incluidas en resúmenes o estadísticas del juego en tiempo real o incluso que permitan la selección de los momentos de mayor impacto para presentar publicidad al espectador.

1.2. Detección de eventos en vídeos de baloncesto

Si nos atenemos al marco de trabajo que definen Min Xu y Ling-Yu Duan en [21], en el caso de vídeos de baloncesto podemos considerar cuatro eventos que definen la estructura del partido, a los que llamaremos por ello *eventos estructurales* y otros cinco *eventos regulares* que tienen gran importancia en el desarrollo del juego y que ocurren con regularidad a lo largo del mismo.

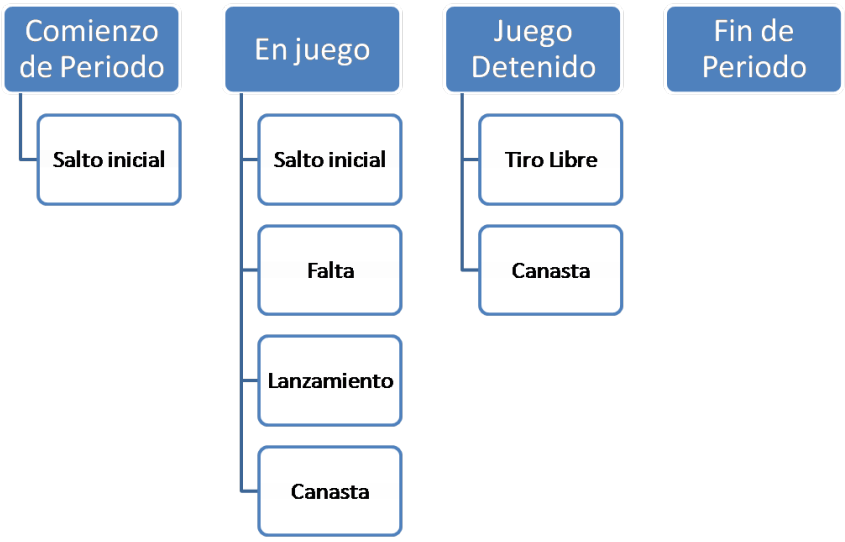

Figura 1.1: Diagrama de eventos estructurales y regulares en partidos de baloncesto.

En la figura 1.1 se puede ver un diagrama de los posibles eventos estructurales y los eventos regulares que pueden ocurrir a lo largo de ellos.

Los eventos estructurales comprenden los siguientes momentos del juego:

- **Comienzo de periodo.** Este evento consiste en el momento de comienzo de cada uno de los cuatro periodos en los que se divide un partido de baloncesto (según las normas FIBA de 10 minutos de duración [4]). En el caso de ser el comienzo del primer periodo coincide además con el evento regular de salto inicial.
- **En juego.** Comprende todo aquel momento en el que el juego no ha sido detenido por el arbitro y el balón se encuentra dentro de los límites del campo de juego.
- **Juego parado.** Comprende todos aquellos momentos en el transcurso de un periodo en el que el juego está parado.
- **Fin de periodo.** Este evento consiste en el momento de final de cualquiera de los periodos.

En el caso de los eventos regulares estos hacen referencia a "jugadas" concretas de especial significado en el juego. Los principales aunque no los únicos son:

- Salto inicial.
- Falta.
- Tiro libre.
- Lanzamiento.
- Canasta.

Para llevar a cabo la detección automática de estos eventos se puede hacer uso de varias características audio-visuales del vídeo.

En el caso del audio, el procesado del mismo en las retransmisiones deportivas contiene gran cantidad de información útil para la detección de eventos debido a la gran correlación que existe entre las acciones de los jugadores y árbitros, los comentarios de los locutores y las respuestas del público. Diversas técnicas de detección de sonidos clave son desarrolladas en los artículos [15], [17], [21], [14].

En cuanto a las características de vídeo, la mayoría de las técnicas se basan en la segmentación de objetos y áreas de interés. En la figura 1.2 se presenta en forma de diagrama de flujo un posible sistema de detección de eventos utilizando exclusivamente características de vídeo. Se ha basado en los marcos de trabajo definidos en los artículos [21], [3] y en la patente [7].

Como se ve, en primer lugar se extraen los fotogramas del vídeo ya sean de uno en uno o en bloques, posteriormente (tras un preprocesado para eliminar algunas aberraciones en el fotograma que puedan causar problemas) se clasifica el fotograma o la secuencia de fotograma valiéndonos de la información de fotogramas anteriores. En función del tipo de plano del fotograma podremos extraer un tipo de información u otra. Estos bloques de procesado según el tipo de plano contienen diversos subsistemas de extracción de características de medio nivel como pueden ser, la detección del campo, detección y seguimiento de jugadores, calibración de la cámara, etc. A partir de la información proporcionada por uno o varios de estos subsistemas podremos detectar algunos de los eventos de alto nivel definidos anteriormente.

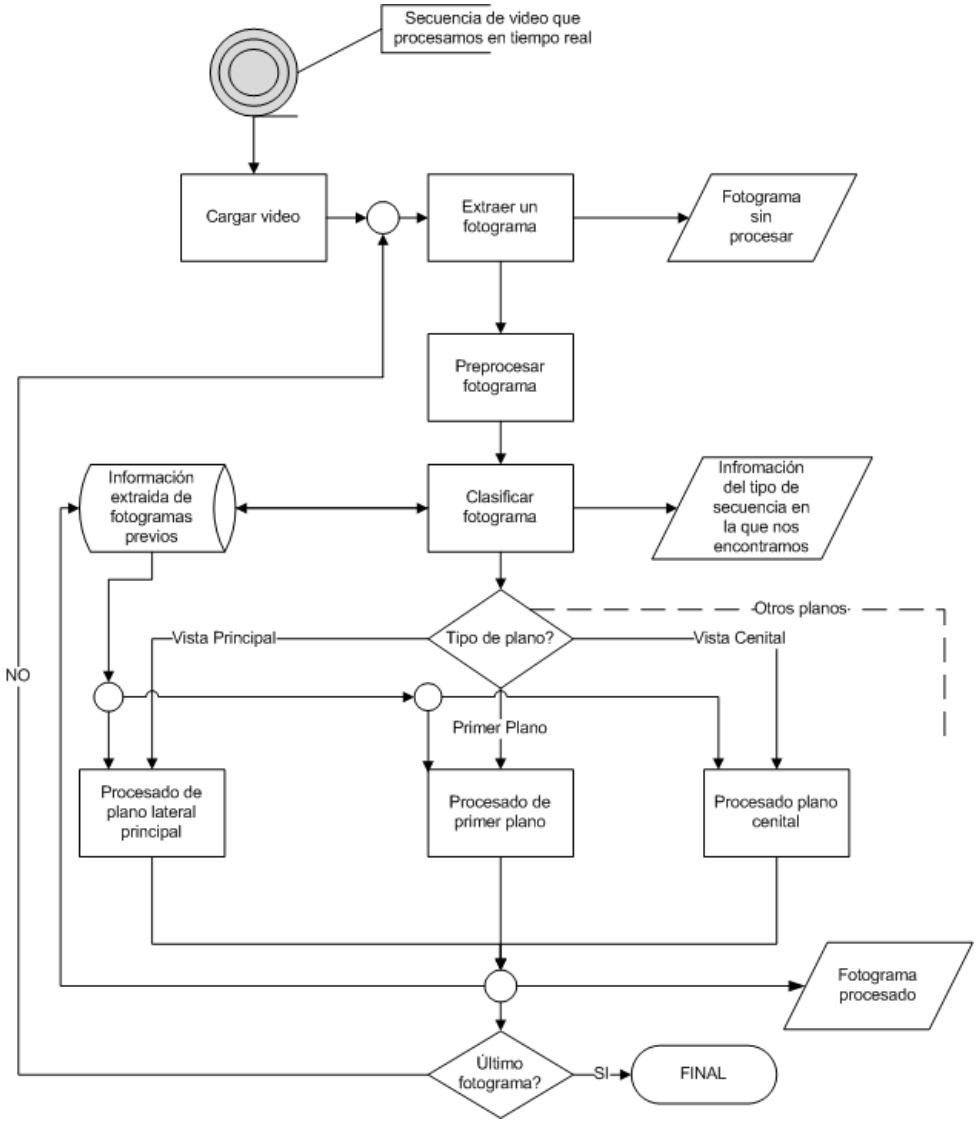

Figura 1.2: Diagrama del marco de trabajo de detección de eventos.

El trabajo desarrollado en el presente proyecto consiste en el estudio e implementación de uno de estos subsistemas, un segmentador de campo de juego para detección de eventos en tiempo real. Este sistema trabaja con un tipo concreto de secuencias de partidos de baloncesto FIBA y es capaz, con un alto porcentaje de acierto, de detectar la zona del campo en dichas secuencias. Esta segmentación que por si sola no proporciona demasiada semántica, podrá utilizarse como base para desarrollar un sistema más complejo que apoyándose en ella, y junto a las otras técnicas, sea capaz de detectar los eventos descritos anteriormente.

1.3. Objetivos del proyecto

En este proyecto, por lo tanto, nos planteamos como objetivo general el estudio, diseño, implementación y prueba de un sistema segmentador en tiempo real de campo de juego en vídeos de baloncesto. Este sistema se encuadrará como se ha dicho dentro de un marco de trabajo de detección de eventos en tiempo real para retransmisiones deportivas. Teniendo en cuenta está aplicación se marcaron los siguientes objetivos concretos:

1. Estudiar el estado del arte referente a la segmentación de campo en vídeos deportivos, haciendo especial énfasis en aquellos estudios centrados en vídeos de baloncesto. Analizar los diferentes problemáticas que presentan vídeos de diferentes deportes a la hora de llevar a cabo la segmentación de campo.

2. Diseñar un sistema que permita realizar segmentación de campo en vídeos de baloncesto de manera robusta basándose en las técnicas estudiadas anteriormente y explorar nuevas aproximaciones que completen los estudios existentes.

3. Implementar dicho sistema haciendo uso de de las capacidades de procesado de imagen y vídeo del paquete *software* Matlab.

4. Realizar pruebas que permitan medir la exactitud y tiempos de ejecución del sistema implementado. Definir bancos de pruebas con diferentes vídeos deportivos que permitan realizar estas medidas. Analizar los resultados obtenidos y extraer conclusiones sobre la calidad, robustez y eficiencia del sistema implementado.

1.4. Medios empleados

Para realizar el trabajo que se desglosará a continuación se ha empleado el *software Matlab* R2009b (versión 7.9.0.529) en su distribución tanto para arquitecturas de 32 bits como de 64 bits. Especialmente se ha utilizado extensivamente el paquete de procesado de imagen *Image Processing Toolbox v6.4* perteneciente a esta distribución de *Matlab*. Asimismo se utilizó para el acceso a los ficheros de vídeo, tal y como se describe en la sección 3.3, la función *mmread* distribuida por Micah Richert bajo licencia BSD a través de *MATLAB Central* [1] .

[1] `http://www.mathworks.com/matlabcentral/fileexchange/8028-mmread`

Además en varios momentos se utilizó para la visualización y la caracterización de los vídeos el reproductor multimedia libre *VLC* en su versión 1.0.5 que forma parte del proyecto VideoLAN que produce software libre multimedia bajo licencia GNU GPS. [2]

El equipo que se utilizó en todo el desarrollo y en las simulaciones es un ordenador portátil *Acer Aspire* con un procesador *Intel Core 2 Duo T7200 (2.0 GHz,667 Mhz FSB, 4 MB L2 cache)* con 1GB de memoria DDR2 y *Windows XP Media Center Edition* como sistema operativo. Adicionalmente para las pruebas de rendimiento de la sección 3.6 se utilizó también un equipo de sobremesa *Medion* con un procesador *Intel Core i5 (CPU 750 @ 2.67 Ghz)* con 6GB de memoria DDR3 y *Windows 7 Home Premium* con la misma distribución de *Matlab*.

1.5. Estructura del documento

El presente documento se organiza de la siguiente forma. En el capítulo 2 se presenta el estado del arte de las técnicas de segmentación de campos de juego en vídeos deportivos, centrándose obviamente en el desarrollo existente en baloncesto, extrayendo las conclusiones que nos permitan iniciar a partir de él el desarrollo de nuestro sistema.

En el capítulo 3 presentaremos la descripción del sistema, de los bancos de pruebas utilizados y del trabajo realizado así como los resultados obtenidos. Ya que el trabajo de realizó de manera evolutiva, realizando un primer prototipo y mejorándolo en sucesivas implementaciones; en esta descripción se hará un análisis crítico las diferentes versiones implementadas, los resultados obtenidos ante un mismo banco de prueba, las mejoras que introducen y sus carencias. Por último se presentarán los resultados de aplicar algunas pruebas más con objeto de estudiar la robustez y eficiencia del sistema.

En el capítulo 4 se presentarán una conclusiones a alto nivel del proyecto y se analizarán las posibles líneas futuras de investigación a seguir que quedan fuera del ámbito de este proyecto.

Por último se incluyen tres anexos. En el anexo A en el que se presenta el presupuesto del proyecto. En el anexo B se presenta la descripción del espacio de color HSV de gran importancia en el trabajo realizado. Finalmente en el anexo C se incluye una descripción de la arquitectura *software* de la aplicación implementada.

[2] `http://www.videolan.org/vlc/`

Capítulo 2

Estado del arte de la segmentación de campo en vídeos deportivos

La detección del campo de juego tiene una gran importancia en el análisis de vídeos deportivos, dado que se puede inferir gran cantidad información semántica del resultado de una segmentación del campo. Dicha segmentación permite clasificar diferentes tipos de fotogramas o planos, identificar objetos de interés como los jugadores o incluso al balón (que estarán probablemente situados en la región de campo de juego), calibrar la cámara para poder obtener las coordenadas reales dentro del campo de dichos objetos a partir de su posición en la imagen o extraer cualquier otra información derivada de la parte del campo en la que nos encontremos: porcentaje de región del fotograma que es de campo, etcétera, que nos permita más adelante detectar eventos de más alto nivel, como por ejemplo estadísticas del juego de cada equipo.

2.1. Aproximaciones a la segmentación de campo. Peculiaridades del baloncesto

Existen diferentes técnicas de segmentación del campo de juego aplicadas a diferentes deportes. Las más estudiadas son aplicadas generalmente sobre deportes de equipo en hierba, y más concretamente fútbol, o sobre tenis; aunque en varios estudios ([8][18][13]) se afirma que funcionan igualmente sobre deportes de interior, como el baloncesto. En realidad los deportes de interior en general (baloncesto, balonmano, fútbol sala,...) y el baloncesto en particular presen-

tan ciertas problemáticas diferentes a los deportes de equipo exterior (fútbol, fútbol americano, hockey hierba, rugby) o tenis a la hora de llevar a cabo la segmentación del campo.

En todos los casos el problema principal es el mismo, identificar la zona de campo dentro de la imagen para posteriormente extraer información semántica de ella y evitar en fases posteriores del procesado el ruido que introduce el público, y las aproximaciones para llevar a cabo esta segmentación son básicamente dos en versiones más o menos sofisticadas:

- Analizar los histogramas de color y detectar la zona de color dominante y considerar el campo como la zona uniforme que corresponda con un rango en torno a ese color.
- Detectar el color de las lineas que limitan el campo y mediante algún algoritmo de estimación, estimar dichas líneas.

Figura 2.1: Fotogramas del plano principal de diferentes deportes: fútbol, tenis, balonmano y baloncesto.

Así por ejemplo en el caso del fútbol se suele utilizar la aproximación de extraer la zona de color dominante dado que el campo siempre será relativamente uniforme, con una tonalidad

similar en todos los campos y además conteniendo todos los elementos destacables del juego (dada la distancia de la cámara en los vídeos de fútbol, los jugadores y la pelota ocupan un área ínfima de campo en comparación con otros deportes) [2] [16] [8] [13] [18]. En cambio en el caso del tenis, dado que como se aprecia en el fotograma de la figura 2.1 el campo suele ser bastante poco uniforme, algunos autores optan por la segunda de las aproximaciones, obteniendo las líneas que limitan el campo ya sea mediante la detección de bordes [22] o del color blanco en la imagen [11]. Dada la escasa bibliografía existente en segmentación de campo aplicada a baloncesto tenemos que tener en cuenta las peculiaridades que presentan los vídeos de este deporte con respecto a los otros de los que si tenemos más información para inferir hasta que punto se pueden extender dichos trabajos al deporte que nos ocupa. Las principales problemáticas de los campos de baloncesto para su segmentación son:

- El campo es de parquet y presenta la textura característica de la madera. Aunque por ello presenta también un color dominante común a todos los campos (cosa que no ocurre por ejemplo en balonmano) éste no es tan claramente uniforme presentando brillos y reflejos.

- Las lineas de campo son en ocasiones poco visibles debido a la oclusión por partes de jugadores y a los brillos del suelo del campo. Por contra, en todos los casos el campo esta rodeado por una zona de un color uniforme, aunque dicho color varía según el campo.

- El plano en deportes de interior es más corto, y por lo tanto los jugadores son notablemente más grandes y pueden tapar gran parte del campo pudiendo llevar a segmentaciones erróneas particularmente en los límites del campo.

- La *zona restringida* [4], no sólo forma parte del campo sino que resulta muy importante para extraer información sobre el partido, pero está pintada de un color diferente al campo y diferente entre distintos campos, por lo que no se puede detectar fácilmente con técnicas de extracción de color dominante. Además otras zonas del campo pueden presentar anuncios publicitarios.

A continuación se expondrán algunas de las técnicas utilizadas en los diversos estudios sobre diferentes deportes, tratando de analizar las ventajas e inconvenientes de las diferentes aproximaciones en su aplicación a nuestro problema en particular.

2.2. Técnicas basadas en detección de la región del color dominante

Como se ha dicho la mayor parte de ellas parten de la misma base, la detección de un color dominante en la distribución de color en un espacio de color determinado. Dado que el campo de juego tiene generalmente un color uniforme que ocupa la mayor parte de la imagen, dicha región dominante indicará que región de la imagen es campo. En lo que difieren los diferentes estudios es en que espacio de color es más apropiado para realizar este análisis. Así por ejemplo el estudio de Yang Liu, Shuqiang Jiang y Qixiang Ye [16] se basa en la detección de picos en el espacio CbCr, en [2] en el espacio HSV (definiendo previamente el rango de posibles valores de hierba), en [13] se trabaja sobre un subespacio de HSI o en [8] Ekin y Murat trabajan a la vez sobre varios modelos de color que proporcionan resultados complementarios que comparan y combinan.

En ninguno de los casos se trabaja sobre el espacio estándar RGB (*red,green,blue*) ya que en él se codifica el color y la luminosidad juntos y por lo tanto no se puede hacer que el sistema sea invariante a cambios de luminosidad, característica necesaria en nuestro caso ya que distintas partes del campo pueden tener muy distinta luminosidad (el campo de baloncesto como ya se ha dicho presenta reflejos y brillos) y se espera que el sistema identifique por igual tanto unas regiones como las otras. En el apéndice B se muestra una somera explicación de en que consiste el espacio de color HSV utilizado en este proyecto, así como las transformaciones para pasar de él a RGB y viceversa, si se desea más información sobre los mismos se recomienda acudir al capítulo 6 de [12].

Tras esta detección de la distribución dominante de color, algunas de las aplicaciones: [16], [13], [18] y [2] utilizan la información de tonalidad dominante para estimar los parámetros de un modelo de mezcla de *gaussianas* (GMM del ingles *Gaussian mixture model*) generalmente con un algoritmo de maximización de la esperanza (EM, *Expectation maximization* [5]). Para entrenar este modelo se utilizan diferentes aproximaciones. En algunos casos se entrena con fotogramas del mismo partido [18], [13]; en otros con fotogramas de distintos partidos elegidos para que el modelo sea suficientemente general [2] y en otros se utiliza una aproximación mixta en la que se usa un conjunto de fotogramas fijo para entrenar el modelo y posteriormente éste se va adaptando mediante algún algoritmo adaptativo a las características del campo y la luminosidad actuales [16].

Posteriormente los diferentes autores realizan diferentes aproximaciones. En [2] se utiliza MAP (*Maximum a posteriori*) para realizar la adaptación de los fotogramas actuales a los modelos ajustados *offline* y así ser capaz de clasificar bloques de píxeles en los dos modelos "campo" y "ruido". En [16] se detecta en que zona del campo de fútbol se encuentran ajustando la clasificación de píxeles a cuatro patrones fijos (lado izquierdo de campo, lado derecho y centro izquierdo y centro derecho) mediante un método de mínimos cuadrados. En todo caso la utilización de GMM tiene como objeto tener un modelado estadístico del campo más exacto que utilizando histogramas y parece lograr mejores resultados [2] [16] [13] pero a cambio de necesitar una etapa de inicialización y entrenamiento.

2.3. Técnicas basadas en detección de lineas de campo

Dado que nosotros buscamos la implementación de un sistema que sea computacionalmente sencillo, autónomo (sin necesidad de un operador) y que funcione sin utilizar ningún conocimiento *a priori* nos resulta más interesante explorar aquellos estudios en los que si bien no se obtiene una representación tan exacta de las estadísticas de color de la imagen, obtienen resultados más eficientemente. Un caso particular es el de la implementación de Chang, Tien y Wu en [3], una muy concreta aplicación de las técnicas de extracción de características que utilizan para el cálculo de estadísticas de alto-nivel del partido que asistan a los preparadores de los equipos de baloncesto. Aunque el estudio parece a priori algo falto de rigor en la exposición de sus resultados, las técnicas aplicadas resultan interesantes. En primer lugar se extrae una máscara a partir del mapa de color dominante, hasta ahí es una técnica parecida a las anteriores, pero posteriormente en ella se identifica el borde al que aplican un algoritmo de tipo RANSAC [23] para detectar las lineas del campo y a partir de ellos segmentarlo, y en este caso calibrar la cámara y posteriormente detectar a los jugadores. Esta idea también es aplicada por Farin y Krabbe en [11] donde se utiliza la detección de lineas de campo en tenis, voleibol y fútbol para llevar a cabo una calibración en tiempo real de la cámara. La diferencia con la implementación anterior es que en este caso no se hace uso de la detección del color dominante sino del color blanco para posteriormente mediante la aplicación de la transformada de Hough detectar líneas y a partir de ahí calibrar la cámara. En la sección 3.3.6 se describe con cierto detalle en que consiste la transformada de Hough (definida en su versión más utilizada por Robert Duda y Peter Hart en [6]) ya que su utilización y estudio constituye un elemento importante del desarrollo de este

proyecto.

2.4. Conclusión. Punto de partida de nuestro trabajo

Excepto en el caso de [11], ninguno de los demás algoritmos encontrados están pensados para funcionar en tiempo real. Por ello, aunque tanto en [2], [16] o [13] se afirma que estas técnicas funcionan mejor que la clasificación basándose en el histograma de color, en nuestra aplicación que deseamos que sea en tiempo real utilizaremos técnicas más sencillas computacionalmente más parecidas al trabajo que se realiza en [3] y en [22] para tenis, combinando por un lado la detección del color dominante con la detección de bordes para extraer las líneas de campo. Se entiende que esta combinación de técnicas además de ser computacionalmente sencillas, resulta la solución con más potencial dadas las peculiaridades de los campos de baloncesto enumeradas anteriormente.

Capítulo 3

Desarrollo y resultados

3.1. Descripción del sistema implementado

Como se ha dicho en la introducción el sistema implementado es un segmentador en tiempo real de campo en vídeos de baloncesto. Básicamente el objetivo es construir un sistema que reciba una secuencia de vídeo determinada extraída de una retransmisión de un partido de baloncesto y que al ir procesando cada fotograma sea capaz de identificar la región de campo en cada uno de los mismos, haciendo uso de la información que pueda extraer de dicho fotograma o de la información obtenida en fotogramas anteriores. Además se desea que esto lo haga de manera eficiente y robusta, pudiendo segmentar el campo de diversos pabellones con canchas de distintas características y de diferentes retransmisiones con posiciones de cámara diferentes.

En todo caso, las secuencias utilizadas son todas de tipo *plano principal lateral* o *plano de campo completo*, tal y como se denominan en [21] y en [15]. En la figura 3.1 se muestran los diferentes tipos de planos típicos en las retransmisiones de baloncesto extraídas de uno de los vídeos de prueba a modo de ejemplo de forma que quede claro a que nos referimos con plano de campo completo.

En los vídeos de retransmisiones deportivas típicamente la mayor parte de la acción del juego sucede durante estos planos de tipo *campo completo*, utilizándose el resto para destacar, normalmente con el tiempo parado, a algún jugador en alguna acción puntual (plano detalle), para repeticiones o en ataques estáticos (plano medio, que puede estar tomado desde uno de los fondos como en la figura 3.1) o para rellenar transiciones y tiempos muertos (plano del público).

A pesar de que existe cierta información que se puede extraer de estos otros tipos de planos,

Figura 3.1: Tipos de plano en baloncesto (de izquierda a derecha y de arriba a abajo : plano de campo completo, plano detalle, plano medio y plano del público).

nos centramos en secuencias de plano de campo completo no sólo por la razón anterior (contienen la mayor parte de la acción del juego y por lo tanto de los eventos) sino porque es en estos tipos de planos en los que la extracción del campo cobra de especial importancia para detectar información de alto nivel. En el resto de tipos de plano, por lo general, o bien la segmentación de campo no nos aporta información relevante sobre el juego o bien el campo directamente no es visible en el plano. Además el subsistema implementado está pensado para formar parte de un sistema de detección de eventos como el descrito en el diagrama de la figura 1.2 y por tanto se puede confiar que estas secuencias hayan sido adecuadamente extraídas utilizando una de las técnicas expuestas en [10], [9] o [21].

Una vez clasificada una secuencia como de campo completo la utilizaremos como entrada de nuestro sistema segmentador. El mismo se haya descrito en el diagrama de bloques de la figura 3.2.

Como se aprecia en el diagrama cada fotograma es preprocesado para optimizar la respuesta del sistema. Posteriormente haciendo uso de la información de fotogramas anteriores se detecta la región de campo como la región de color dominante y se genera una máscara a partir de dicha información. A partir de esta máscara extraemos las fronteras del campo. Por último a partir de esta frontera del campo generamos una máscara final que separe la zona del campo de la zona exterior además de cierta información de alto nivel (en que zona del campo nos encontramos) y

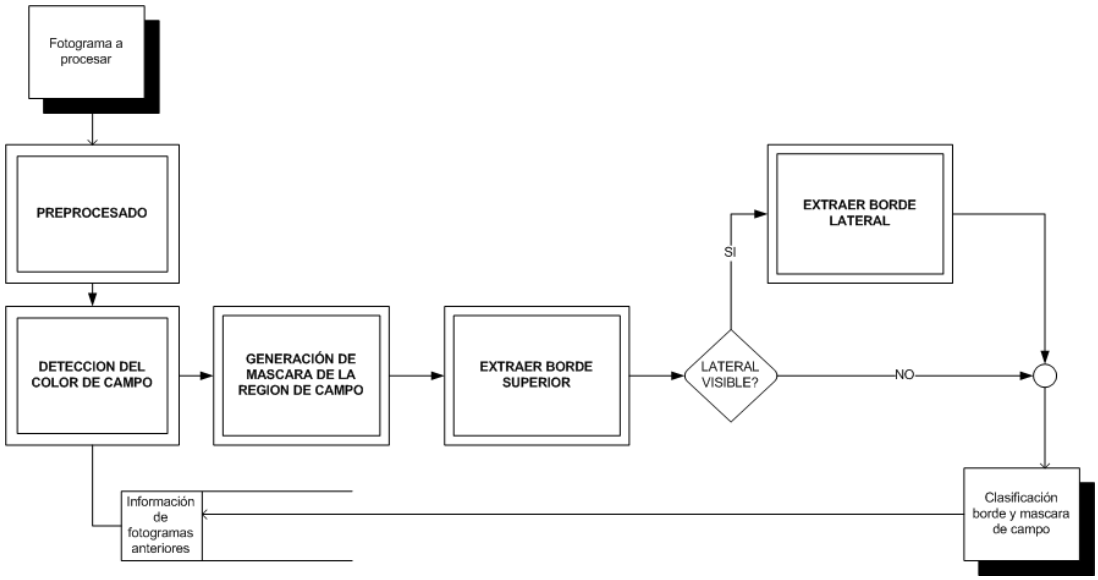

Figura 3.2: Diagrama de bloques del sistema segmentador de campo en videos de baloncesto.

una medida cuantitativa de la calidad de la segmentación, esta información la podremos utilizar en el siguiente fotograma y volver iterar el sistema. En la figura 3.3 se representa un ejemplo del resultado que se espera del sistema en cada iteración. Como se ve la segmentación que se hace consiste en una delimitación del borde superior del campo y del borde lateral (en el caso de que este sea visible) y no del borde inferior. Se decidió esta estrategia por el hecho de que el borde inferior no sólo no se haya siempre presente en la imagen sino que además su segmentación no aporta mucha información, ya que existen pocos o ningún elemento ajeno al juego que se pueda eliminar con la detección de este borde.[1].

En las siguientes secciones profundizaremos en la implementación de cada uno de las partes del sistema. En la sección 3.2 presentaremos el banco de pruebas utilizado para diseñar y evaluar el sistema, así como las pruebas realizadas. En la siguiente sección, 3.3, presentaremos la primera solución implementada, los resultados obtenidos y los problemas que presenta. En las siguientes secciones, 3.4 y 3.5 mostraremos de manera progresiva las sucesivas implementaciones realizadas,

[1] Encontrar el borde inferior y más concretamente la esquina inferior (derecha o izquierda) del campo puede resultar interesante si en un futuro se utilizase este sistema para calibración de cámara, tal y como se hace con tenis, voleibol y fútbol en [11]. Se podría utilizar junto con la esquina superior (derecha o izquierda) para calcular la proyección del campo realizada por la cámara a partir de las dimensiones del mismo, pero analizando los resultados del sistema y las pocas veces que dicha esquina aparece en las imágenes quizás esta calibración se realizara mejor utilizando otro punto identificable como los límites de la *zona restringida*.

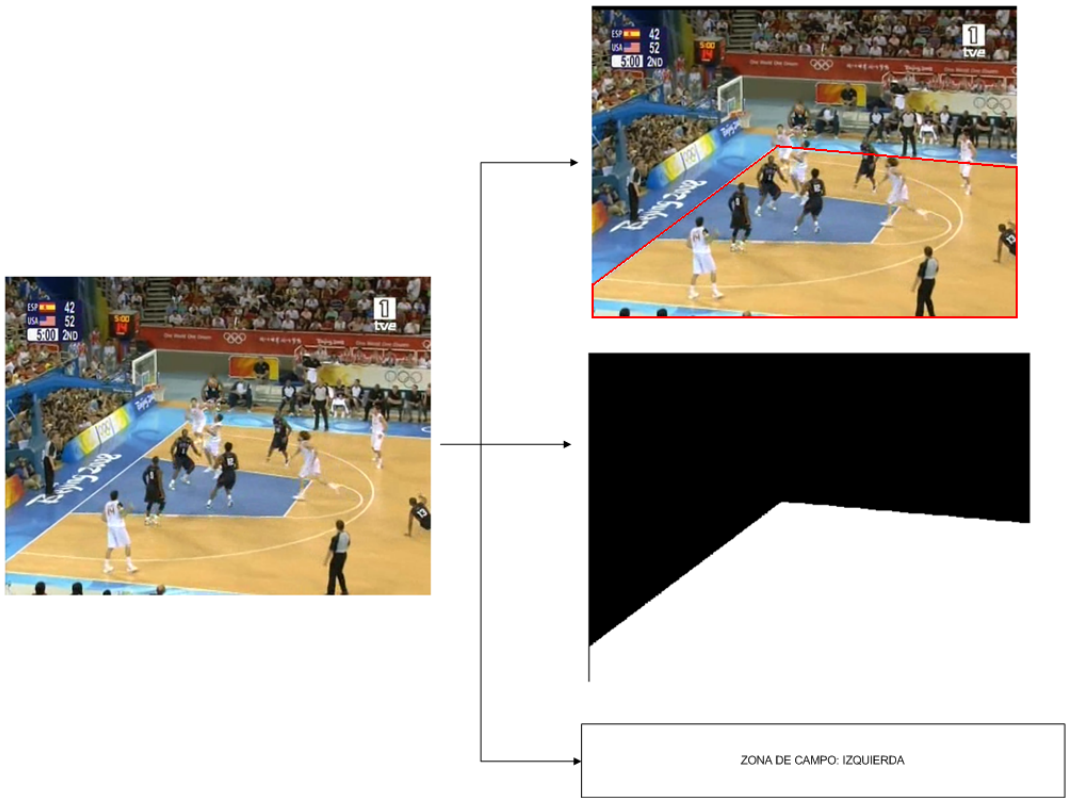

Figura 3.3: Ejemplo de resultado esperado del sistema. A la izquierda el fotograma que usamos a la entrada; a la derecha y de arriba a abajo, ese mismo fotograma representado junto con el borde de campo, la máscara generada a partir de dicho borde y la información del lado del campo en el que nos encontramos.

la mejoras añadidas para solventan los problemas de versiones anteriores y los resultados a las pruebas realizadas. Finalmente en la sección 3.6 realizaremos sobre la versión final otra batería de pruebas para obtener unos resultados fiables de validez, robustez y eficiencia del sistema.

3.2. Descripción del banco de pruebas

Como en cualquier otro sistema de procesamiento es necesario definir las métricas y los tests que nos permitan determinar la validez del mismo, así como otras medidas como el tiempo de cómputo empleado en las simulaciones, etcétera. Con objeto de realizar estas medidas de manera robusta se definieron dos bancos de pruebas diferentes, un banco de pruebas de desarrollo y un banco de evaluación. El primer banco se utilizó a lo largo de todo el diseño y codificación del sistema para evaluar su funcionamiento, detectar los problemas de las distintas implementaciones y poderlas mejorar. El segundo banco de pruebas solamente se utilizó una vez implementada la solución final para medir su calidad y eficiencia de una manera robusta. Para mejorar esta medida de robustez, los fotogramas y secuencias del banco de pruebas de desarrollo y del banco de pruebas de evaluación pertenecen a vídeos de retransmisiones diferentes.

Todos los partidos utilizados en los bancos de pruebas son partidos según las reglas de *Baloncesto FIBA*, definidas por la Federación Internacional de Baloncesto [4], esto excluye partidos de la liga estadounidense de baloncesto profesional (NBA *National Basketball Association*). La razón de utilizar únicamente partidos de *baloncesto FIBA* se tomó en los primeros estadios del diseño del sistema debido a las importantes diferencias existentes en las medidas y forma de los campos de ambas corrientes de baloncesto, por lo que se optó por diseñar un sistema que funcionara de forma robusta en un tipo de partidos. En la figura 3.4 se observan estas diferencias en fotogramas de dos partidos FIBA y dos partidos NBA.

Algunas diferencias son que los campos tienen unas dimensiones distintas, las zonas restringidas tienen diferente forma, en NBA no siempre está pintada uniformemente, normalmente en NBA el plano es más corto que en baloncesto FIBA, etcétera.

En todo caso en la sección 3.6.3 se muestran algunas pruebas realizadas sobre fotogramas de partidos NBA con el objeto de evaluar las posibilidades de ampliar el rango de partidos utilizables en futuras mejoras del sistema.

En total se utilizaron vídeos de fragmentos o retransmisiones completas de 10 partidos de baloncesto diferentes. Los vídeos utilizados y sus características se muestran en las siguientes tablas 3.1 y 3.2:

El banco de pruebas de desarrollo está formado por dos tests que hacen uso de fotogramas y secuencias de su grupo de vídeos, estos fotogramas y secuencias han sido seleccionados de forma que muestren diferentes jugadas y zonas del campo.

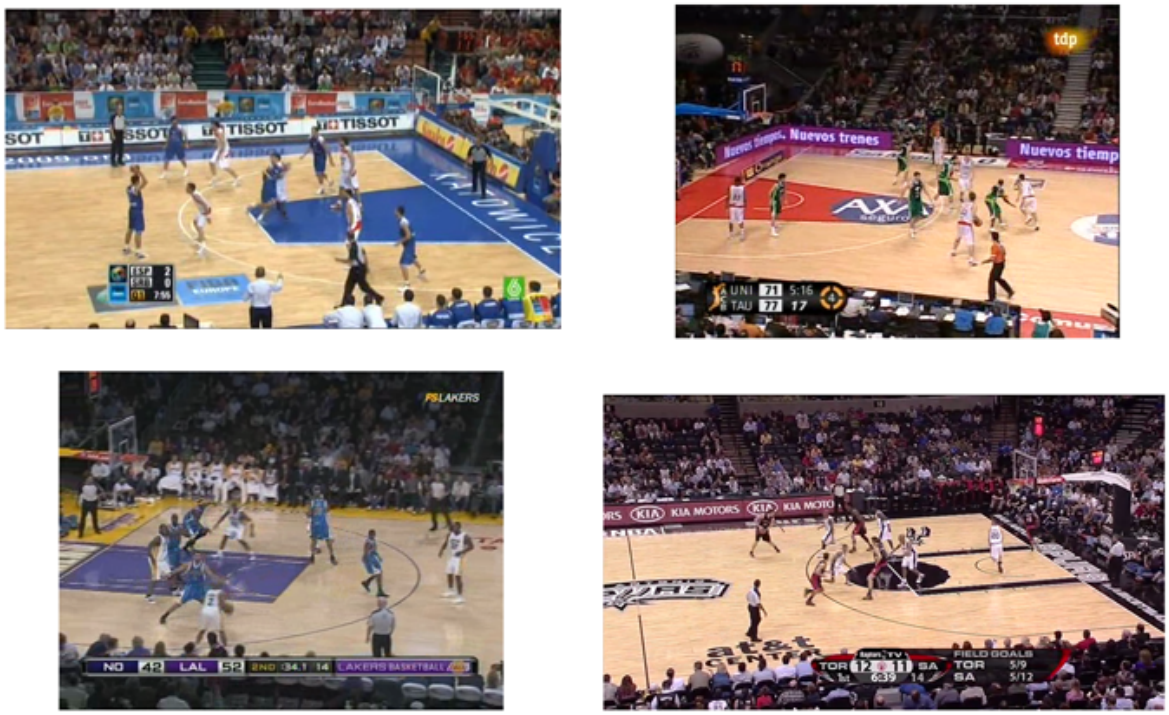

Figura 3.4: Comparativa de fotogramas de partidos FIBA (arriba) con partidos NBA (abajo).

- El primer test está compuesto por 5 fotogramas seleccionados de cada vídeo. Por lo que tendremos 25 fotogramas de diferentes momentos del juego.

- El segundo test está compuesto por 5 secuencias de 1 segundo de duración cada una, íntegramente de plano de campo completo, una de cada vídeo. Dado que todos los vídeos tienen una tasa de 25 fotogramas por segundo. Este segundo test tendrá 125 fotogramas en cada banco de pruebas.

El banco de pruebas de evaluación sólo tendrá el segundo test, es decir 5 secuencias de vídeo de 1 segundo de duración, íntegramente de plano de campo completo, una de cada vídeo. Este test es el que nos permite evaluar el sistema completo, mientras con el otro (que esta formado por fotogramas sueltos) no podemos aprovecharnos de la correlación existente entre fotogramas consecutivos.

En estos tipos de sistemas es difícil definir una métrica objetiva que permita cuantificar la calidad de la segmentación. En consecuencia en estas dos pruebas, coincidiendo con algunos ar-

Vídeos Utilizados (Pruebas de desarrollo)					
	Competición	**Partido**	**Fecha**	**CoDec**	**Resolución**
1	Eurobasket 2009	Final masculina: España-Servia	20 Sep 2009	XVID	640x368
2	Juegos Olímpicos de Beijing 2008	Final masculina: España-USA	16 Ago 2008	XVID	640x480
3	Copa del Rey de Baloncesto 2009	Final: Tau Cerámica Baskonia - Unicaja Málaga	22 Feb 2009	XVID	512x384
4	Copa del Rey de Baloncesto 2009	Cuartos de Final: Regal FC Barcelona - Real Madrid CF	19 Feb 2009	XVID	512x384
5	Euroliga 2009	Cuartos de Final (3er Partido): Olympiacos BC - Real Madrid CF	31 Mar 2009	XVID	513x384

Tabla 3.1: Vídeos utilizados en el banco de desarrollo.

Vídeos Utilizados (Pruebas de evaluación)					
	Competición	**Partido**	**Fecha**	**CoDec**	**Resolución**
1	Liga ACB 2008-2009	Jornada 33: CB Granada - Unicaja Malaga	30 Abr 2009	XVID	640x360
2	Euroliga 2010	Top 16: Olympiacos BC - Caja Laboral Baskonia	04 Mar 2010	H264	720x576
3	Eurocup 2010	Final: Alba Berlin - Power Electronics Valencia	18 Abr 2009	AVC1	640x480
4	Euroliga 2010	Final: Regal FC Barcelona - Olympiacos BC	May 2010	AVC1	704x384
5	Euroliga 2010	Top 16: Maroussi BC - Regal FC Barcelona	28 Ene 2010	AVC1	576x432

Tabla 3.2: Vídeos utilizados en el banco de pruebas de evaluación.

tículos de la bibliografía [18] [2] [16], utilizaremos como métrica la tasa de fotogramas con el campo correctamente segmentado, considerando como correctamente segmentados aquellos que subjetivamente consideremos que separan con suficiente exactitud el campo. Aunque esta es una

métrica subjetiva, parcialmente dependiente del juicio del observador, creemos que es suficientemente general y fiable para una primera aproximación. Se sugiere por tanto para una futura investigación la posibilidades de utilizar alguna de las magnitudes usadas en el sistema como métrica objetiva de mayor precisión. En la figura 3.5 se muestran algunos ejemplos de segmentaciones que se consideran correctas (fila superior) y de algunas que se consideran incorrectas (fila inferior).

Figura 3.5: Ejemplos de segmentaciones correctas (fila superior) y segmentaciones erróneas (fila inferior).

Como se ve en los fotogramas de la última columna algunos casos pueden resultar ambiguos, por lo que en ejemplos como este hay que tomar decisiones en cuanto a la importancia que puede tener un error en la detección de uno de los bordes del campo frente a otro, pensando en su utilidad a la hora de detectar eventos. En este caso concreto por ejemplo es obvio que la detección del fondo del campo aunque no sea exacta en la figura de arriba a la derecha proporciona una segmentación bastante buena del campo; en el caso de la fila inferior el fondo del campo no es ni siquiera detectado perdiendo por tanto información como por ejemplo en que parte del campo en la que nos hayamos, donde termina el mismo, donde está situada la *zona restringida* [4] o la canasta, etcétera.

3.3. Versión Inicial. Versión 0.1

En la presente sección describiremos la primera solución implementada. Para ello analizaremos los distintos bloques funcionales del diagrama 3.2.

3.3.1. Extracción de fotogramas

Antes de describir las funciones de nuestro sistema es necesario comentar como se cargan los fotogramas y las secuencias de vídeo en Matlab para usarlas como entradas del mismo.

Como se explica en 3.2 existen dos tipos de pruebas en los dos bancos: uno que utiliza fotogramas individuales capturados de los diferentes vídeos y el segundo que utiliza secuencias completas de 1 seg de los diferentes vídeos.

En el primer caso simplemente tenemos los fotogramas capturados mediante el reproductor multimedia *VLC Media Player*, almacenados en formato PNG (*Portable Network Graphics*). Para cargarlos en Matlab tenemos que utilizar la función *imread* del *Image Processing Toolbox* obteniendo la representación RGB de la imagen, una matriz de tres dimensiones: MxNx3, donde M es el numero de píxeles en vertical, N en horizontal y 3 son las componentes que tenemos por píxel (R,G,B). Los elementos de la matriz son de tipo *uint8* con un valor entre 0 y 255 (al tener profundidad de color de 24 bits, 8 por componente), en alguna de las funciones utilizadas necesitaremos que la información de la imagen esté en formato *double* por lo que llevamos a cabo una conversión mediante la función *im2double*. Tras esta operación tendremos la representación RGB de la imagen como una matriz MxNx3 con elementos de valor entre 0 y 1, que será la entrada a nuestro sistema.

En el segundo caso se pueden llevar a cabo dos aproximaciones. La más directa es cargar el vídeo utilizando la función *mmread* compartida por Micah Richert bajo licencia BSD a través de *MATLAB Central*. Se usó esta función en lugar de *mmreader* incluida en el *Image Processing Toolbox* ya que esta tenía problemas al decodificar alguno de los vídeos utilizados, problemas que no aparecían con *mmread*. A esta función hay que especificarle una serie de parámetros como son el *path* del fichero a leer y los fotogramas que se desean extraer, y se obtiene como resultado una estructura de datos con los distintos fotogramas y la información sobre ellos (campo *frames.cdata*) en el mismo formato que se obtiene al usar *imread* por lo que realizando la conversión de *uint8* a *double* mediante *im2double* obtenemos las mismas matrices de entrada que en el caso anterior. Esta aproximación tiene un problema: la función *mmread* es extremadamente lenta, tardando

varios minutos en extraer 1 segundo de vídeo. Esto llevó a que se decidiera extraer los fotogramas de las secuencias de vídeo una sola vez, guardarlos en ficheros de formato PNG mediante la función *imwrite* y posteriormente cargarlos en el momento que sea necesario a través de *imread* como se explica en el caso anterior.

Es necesario destacar que este tiempo de carga de fotogramas no se contabilizará en las medidas de tiempo de cómputo del sistema, ya que en un entorno real es de esperar que se utilicen vídeos no comprimidos (o no tan comprimidos) y que exista un sistema rápido, quizás implementado en *Hardware*, que se encargue de proveer al sistema de fotogramas en un tiempo despreciable.

3.3.2. Preprocesado del fotograma

Una vez que tenemos el fotograma cargado en Matlab y antes de empezar a tratar de segmentar el campo de baloncesto, podemos llevar a cabo alguna operación de preprocesado sobre el mismo para mejorar la respuesta del sistema o prevenir algunos problemas que puedan ser detectados.

Existen numerosas técnicas de realce de imagen (ver capítulo *image enhancement* en [12]) pero en el sistema implementado no se encontró la necesidad de aplicar ninguna de estas técnicas, por lo que el bloque de preprocesado solo realiza 2 operaciones muy sencillas:

- En primer lugar se eliminaron unos pocos píxeles del borde de la imagen que pueden presentar un artefacto debido a la compresión del vídeo en forma de una franja negra de 2 o 4 píxeles que puede resultar molesto (ver figura 3.6).
- En segundo lugar se fijaron a color negro unos cuantos píxeles en la parte superior de la imagen. La razón de esta modificación radica en la forma en la que se genera la máscara de color y será explicada en la sección 3.3.4.

3.3.3. Detección del color de campo

Basándonos en las conclusiones extraídas en las sección 2.4 de los estudios analizados se decidió que el primer paso para llevar a cabo la segmentación del campo en vídeos de baloncesto debía ser la detección de la zona de color dominante. La idea consiste en detectar los píxeles de la imagen que presenten una tonalidad similar, dentro de un rango, a la tonalidad dominante de la imagen.

Figura 3.6: Fotograma en el que se aprecia un artefacto de compresión como unas estrechas franjas negras en los laterales.

Esta detección se realiza en el espacio de color HSV descrito en el apéndice B, concretamente sobre la componente H. La componente H (*Hue*, en castellano matiz) es invariante a cambios de saturación de color o brillo, ya que estas se codifican en las otras dos componentes, S y V respectivamente, por lo que resulta la adecuada para identificar zonas de la imagen que presentan la misma tonalidad aunque están sometidas a distintas condiciones de luminosidad. Por lo que se realiza un transformación del fotograma que es una imagen RGB a HSV siguiendo las ecuaciones presentes en el apéndice B. Esta transformación nos genera a partir de la matriz tridimensional que es el fotograma en RGB, otra matriz de la misma dimensión con las componentes H, S y V. Si extraemos la componente que nos interesa, componente H, entonces tendremos una matriz de tamaño: numero de píxeles la imagen de alto , número de píxeles de la imagen de ancho, con elementos que varían entre 0 y 1. En la siguiente figura (figura 3.7) se ve la representación de la componente H de tres fotogramas distintos, en las tres se ha fijado S=0.5 y V=1.

Sabemos, viendo el cono de la figura B.1 que la componente H es una componente angular donde 0 y 360^{o} (0 y 1) corresponde al rojo y todos los valores intermedios codifican las distintas tonalidades. Por lo tanto al necesitar unas tonalidades en el entorno de los marrones los valores de H donde debería estar en el primer sextante (menos que 60^{o}) es decir aproximadamente $H < 0{,}166$ (ver figura 3.8).

Se analizaron los histogramas de H de diversos fotogramas (los 25 fotogramas del banco de test de desarrollo 1) con objeto de comprobar esta hipótesis y fijar un rango de H en el que etiquetar como “campo” en la imagen. Como se ve en la figura 3.9 el histograma tiene un pico

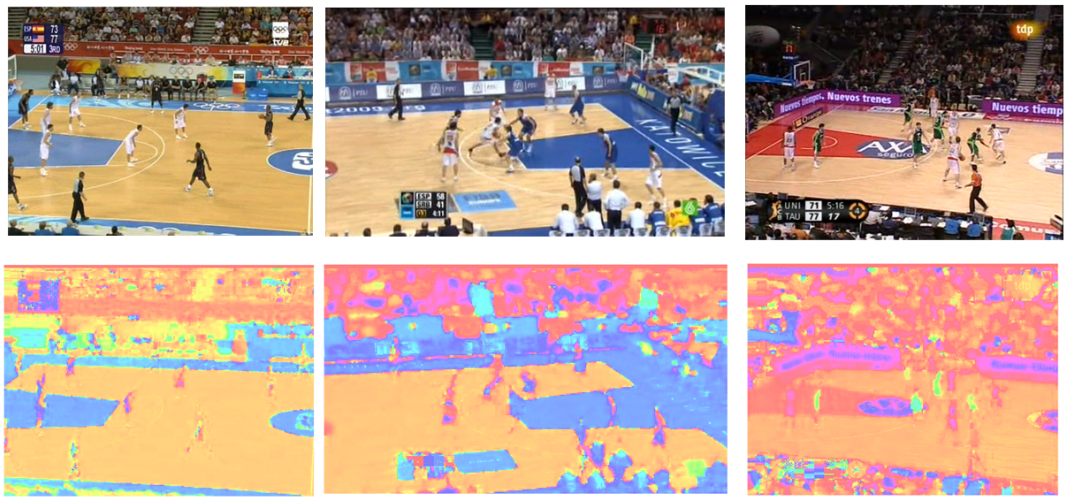

Figura 3.7: Componente H de tres fotogramas distintos (S=0.5, V=1).

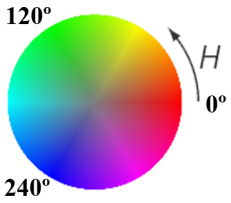

Figura 3.8: Circulo HSV de V=1 (extraída de [19]).

muy importante en torno a H=0.1 (36º), que corresponde al color dominante de la imagen, que es el color del campo. En el mapa de color de la figura 3.10 se observa que efectivamente que este valor de H corresponde a un matiz marrón.

Por ello para identificar que píxeles son candidatos a ser píxeles del campo simplemente tenemos que delimitar un rango de H que comprenda estos valores dominantes de H, encontrar que píxeles presentan una tonalidad en ese rango y etiquetarlos como píxeles de campo. La identificación de los colores de campo no se hace en realidad con un rango sino con dos rangos:

- El primero es un rango fijo en H, que constituye el primer parámetro importante del sistema, en la zona en la que sabemos que con certeza se encontrará el color del campo. Este rango es bastante amplio y su único objetivo es dejar fuera del mismo a aquellos otros posibles colores con una cierta preponderancia (por ejemplo el color de las zonas y de las bandas) y sobre todo a la respuesta H=0 que ocurre cuando los píxeles son negros (situación común en la parte de las gradas) a la hora de buscar la H dominante. Este rango

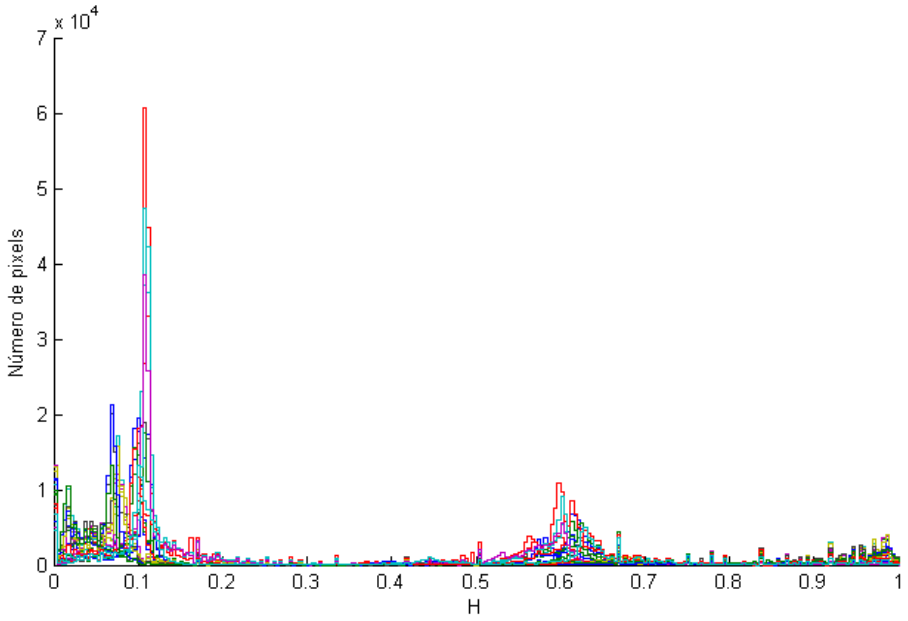

Figura 3.9: Histograma de H de los fotogramas del bando de test de desarrollo 1.

Figura 3.10: Mapa de color H=0.1, extraído del selector de color de Adobe Photoshop CS.

podría ser ajustado si se deseara utilizar en otro deporte donde el suelo no fuera de madera, por ejemplo poniéndolo en torno al verde en partidos sobre hierba, y en esta versión tiene el siguiente valor:

$$R_h1 = [0{,}03\ ,\ 0{,}15] \tag{3.1}$$

- El segundo es un rango variable que constituye lo que verdaderamente vamos a considerar color de campo. Este rango se forma con el máximo y mínimo valor de H encontrado entre las H dominantes en el rango anterior. Se entiende H dominantes los 10 valores de H con

más ocurrencias dentro de este rango. A estos valores dominantes máximo y mínimo se les añade una tolerancia para hacer el rango un poco menos estricto. Esta tolerancia se ajusto empíricamente a un valor ±0,01.

$$\begin{gathered} R_h2 = [Min(H_{dom}) - Tolerance_H, Max(H_{dom}) + Tolerance_H] \\ Tolerance_H = 0{,}01 \end{gathered} \tag{3.2}$$

En Matlab la función utilizada para llevar a cabo el paso de RGB a HSV tal y como se explica en el apéndice B es *rgb2hsv*. Los histogramas son generados con la función de Matlab *imhist* que divide el rango de valores de los píxeles de la imagen en 256 clases, por lo que tendremos una resolución en H de 0.004 (1.44º).

Una vez que se a delimitado el rango de matiz para considerar un píxel de campo, lo tenemos que etiquetar como tal, esto se hace mediante la generación de una máscara.

3.3.4. Generación de máscara de región de campo

En el contexto del procesado de imagen una *máscara* es algo muy similar al concepto informático de máscara de bits. La idea es asignar a cada píxel un valor binario [0,1], en nuestro caso dependiendo de si lo consideramos píxel de campo o no, pudiendo así identificar la zona del campo y realizar así operaciones sobre toda la imagen que sólo apliquen a la zona del campo, a esto es a lo que se le llama enmascarado.

Para generar la máscara de nuestros fotogramas que diferencie los píxeles con color de campo del resto simplemente generamos una matriz 2D de las mismas dimensiones que la imagen. Y le asignamos un 1 en caso de que la componente H de ese píxel este dentro del rango R_h2 definido anteriormente y 0 en caso contrario. Así al aplicarla sobre la imagen con una operación lógica AND (o equivalentemente una multiplicación) obtenemos una imagen exactamente como la que teníamos pero con los píxeles que no consideramos de campo puestos a negro. En la figura 3.11 se ve un ejemplo de la acción de enmascarado. A partir del fotograma de la imagen hemos extraído su máscara con los rangos de valores de H definidos anteriormente. Cuando aplicamos esta máscara mediante una multiplicación de la misma con las tres componentes de la imagen (R, G y B) obtenemos la imagen que teníamos pero con los píxeles anotados con 0 en la máscara puestos a negro.

Como se ve en la imagen, el sistema de detección de color dominante tiene una alta tasa de

Figura 3.11: Ejemplo de enmascarado. Al multiplicar el fotograma por la máscara, sólo los píxeles del fotograma etiquetados con 1 en la máscara se conservan en el resultado.

acierto en la identificación de la zona del campo (pocos errores de tipo 2) aunque existe una amplia zona del público de color dentro del rango (y por lo tanto numerosos errores de tipo 1) [2] [3].

No debemos olvidar en todo caso que nuestro objetivo es segmentar la zona de campo del resto de la imagen y que por lo tanto no nos interesa que los elementos que están sobre el campo pero que son de otro color (jugadores, carteles de publicidad, etcétera) se identifiquen como que no pertenecen al campo. Por ello llevamos a cabo una operación con la máscara que se denomina *filling* (rellenado). Este rellenado es en una operación morfológica que se realiza sobre una imagen binaria (como nuestra máscara) y que rellena los huecos de la misma. Los huecos son los conjuntos de píxeles de fondo (píxeles negros) que no están conectados con el borde de la imagen (ver figura 3.12).

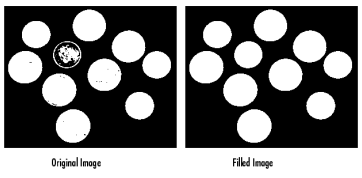

Figura 3.12: Ejemplo de rellenado extraído de la documentación de *imfill* de Matlab.

Como se ve aquellos píxeles negros que no estaban conectados con el borde de la imagen, han sido *rellenados* de blanco. Si aplicamos esta operación sobre nuestra máscara aquellos elementos

[2] *Error de tipo 1:* se entiende por error de tipo 1 el decidir que un píxel es píxel de campo cuando en realidad no lo es.

[3] *Error de tipo 2:* Se entiende por error de tipo 2 interpretar un píxel como no de campo y que si lo sea.

que no se identificaron como de campo pero que se hayan rodeados de campo por todos lados como los jugadores, las carteles pegados en el campo, etcétera; se marcarán como zona de campo. Esto, como se ha dicho en el párrafo anterior, nos interesa para nuestra aplicación pero en seguida se aprecian las limitaciones de esta técnica ya que en el caso en el que los elementos que querríamos eliminar aparezcan solapados unos con otros y con el fondo al estar conectados con el mismo no se eliminarán con este rellenado (ver segunda fila de la figura 3.13).

En cualquier caso, este rellenado no nos soluciona el mayor problema que tenemos: no nos elimina los píxeles mal interpretados como píxeles de campo fuera del mismo, es decir nos permite eliminar (con limitaciones) los errores de tipo 2, pero no los de tipo 1 que como dijimos en nuestro caso son más numerosos. Para eliminar estos tendríamos que rellenar no los píxeles de fondo (píxeles negros) inconexos sino los píxeles blancos inconexos. Esto es sencillo, simplemente tenemos que aplicar la operación de rellenado anterior a la máscara negada (1 - máscara). De esta forma lograremos eliminar todas las islas de falsas regiones de campo que aparecen en la imagen entre el público (ver tercera fila de la figura 3.13).

En este punto es donde cobra sentido el preprocesado explicado en la sección 3.3.2 en el que añadimos artificialmente una franja negra en el borde superior. Existen casos en los que una o varias zonas relativamente grandes de falsos píxeles de campo están conectadas con el borde superior y por lo tanto la operación de rellenado no las eliminaría. Al añadir una franja negra en el borde superior este problema se elimina y esto se puede hacer ya que en los planos que estamos trabajando la parte superior del fotograma nunca es de campo. En la figura 3.14 se muestran las máscaras obtenidas para un mismo fotograma, con y sin preprocesado. Se observa claramente que si no añadimos esta franja negra no somos capaces de eliminar los errores que aparecen en la parte superior de la imagen y que pueden provocar que el sistema no funcione correctamente.

La máscara que hemos obtenido delimita en general de manera bastante correcta la zona del campo pero sin incluir algunos de los elementos que se encuentran sobre el mismo: jugadores, pelota o elementos pintados. Lo que de ahora en adelante trataremos de conseguir es, a partir de esta máscara, encontrar los limites de campo que mejor ajusten esta máscara. Para ello trataremos de encontrar los bordes de la máscara y a partir de ahí de una forma similar a como se hace en [3] y [22] detectar las líneas de campo como direcciones dominantes de entre estos bordes.

Ya en este punto se observa alguno de los efectos secundarios que tiene el aplicar el rellenado de la máscara (concretamente el rellenado de los huecos blancos). En la máscara de la esquina

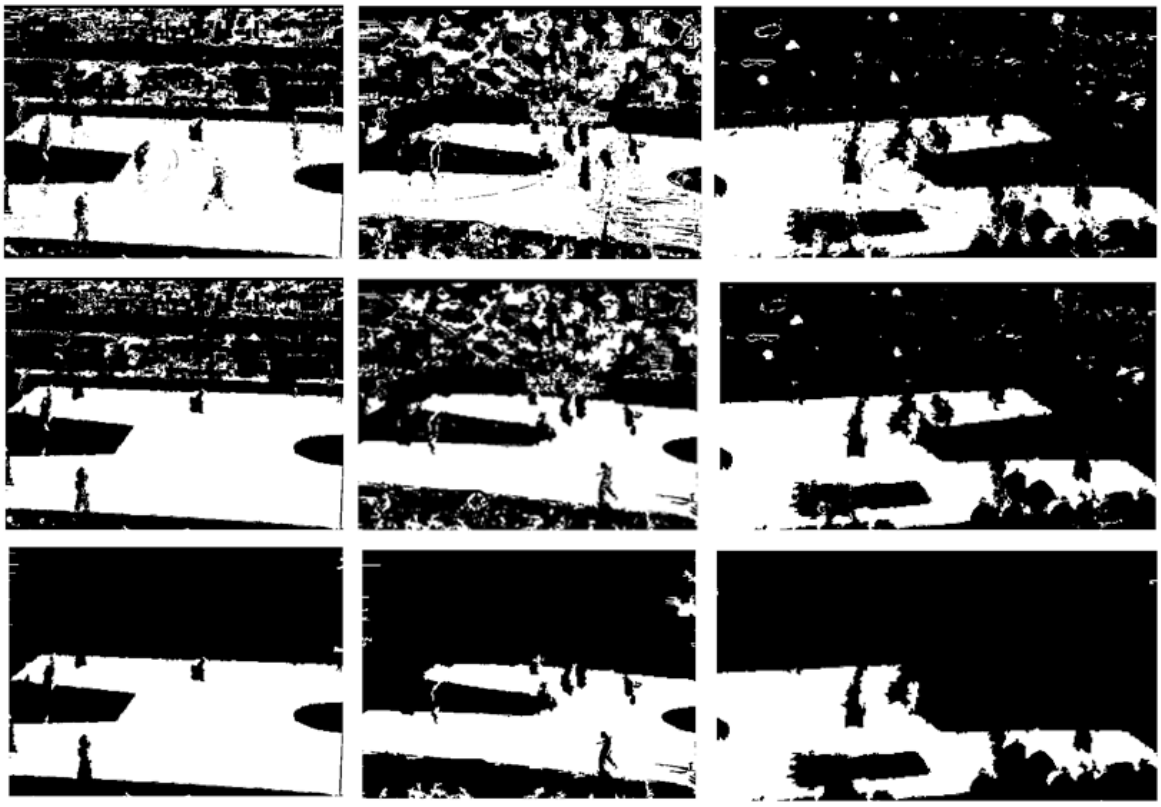

Figura 3.13: Ejemplo de rellenado de máscaras de campo (En la primera fila está la máscara sin rellenar, en la segunda la máscara con los huecos negros rellenos y la tercera con los huecos blancos rellenos).

inferior derecha de la figura 3.13 se ve como la situación de los jugadores en el campo provoca errores de tipo 2 al llevar a cabo este rellenado, parte del campo se considera erróneamente como no campo. En todo caso hay que tener en cuenta que de esta máscara todavía podemos extraer los bordes y a partir de ellos las líneas que limitan al campo, pero habrá que tener en mente esta situación a la hora de decidir cuales son los bordes que mejor ajustan el campo y es posible que provoque que un borde lateral visible en la imagen no sea detectado.

Este rellenado implementado tal y como se ve en las figuras anteriores se realiza en Matlab mediante la función *imfill* presente en el *Image Processing Toolbox*.

Figura 3.14: Justificación del preprocesado. A la izquierda tenemos la máscara obtenida al final del proceso si no preprocesamos, a la derecha llevando a cabo el preprocesado.

3.3.5. Extracción de bordes de la máscara

Después de la extracción de la máscara, como se ve en el diagrama de bloques de la figura 3.2, detectaremos los limites del campo en dos pasos: en primer lugar el borde superior y a partir de la información obtenida con esta detección, si es necesario, trataremos de detectar el borde lateral. En ambos casos la herramienta utilizada es la misma la transformada de Hough, aunque cambian algunos parámetros. Antes de explicar en que consiste esta herramienta tenemos que extraer los bordes de la máscara que se utilizaran como entrada de la misma.

En esta primera aproximación utilizaremos la función *edge* del *Image Processing Toolbox*. Si no se le especifica ningún parámetro utiliza la aproximación a la derivada de Sobel para encontrar los bordes, devolviendo como tales aquellos puntos donde el gradiente de la imagen es máximo. No profundizaremos en los operadores de Sobel, ya que como se verá más adelante esta solución fue sustituida en la versión final, pero si se desea más información se puede encontrar en [12]. Obviamente si nos fijaremos en los resultados obtenidos al aplicar estos filtros a nuestra máscara que presentamos en la figura 3.15.

Como se ve la función es muy precisa a la hora de encontrar los bordes de nuestra máscara. Se podría decir que es demasiado precisa para nuestra aplicación. Como se verá más adelante, el hecho de que estos bordes no sean lisos sino muy rugosos nos va a provocar una respuesta menos clara de la transformada de Hough, ya que las direcciones son menos claras. Otro problema que aparece en algunos partidos es la aparición de bordes paralelos al borde del campo detrás de la

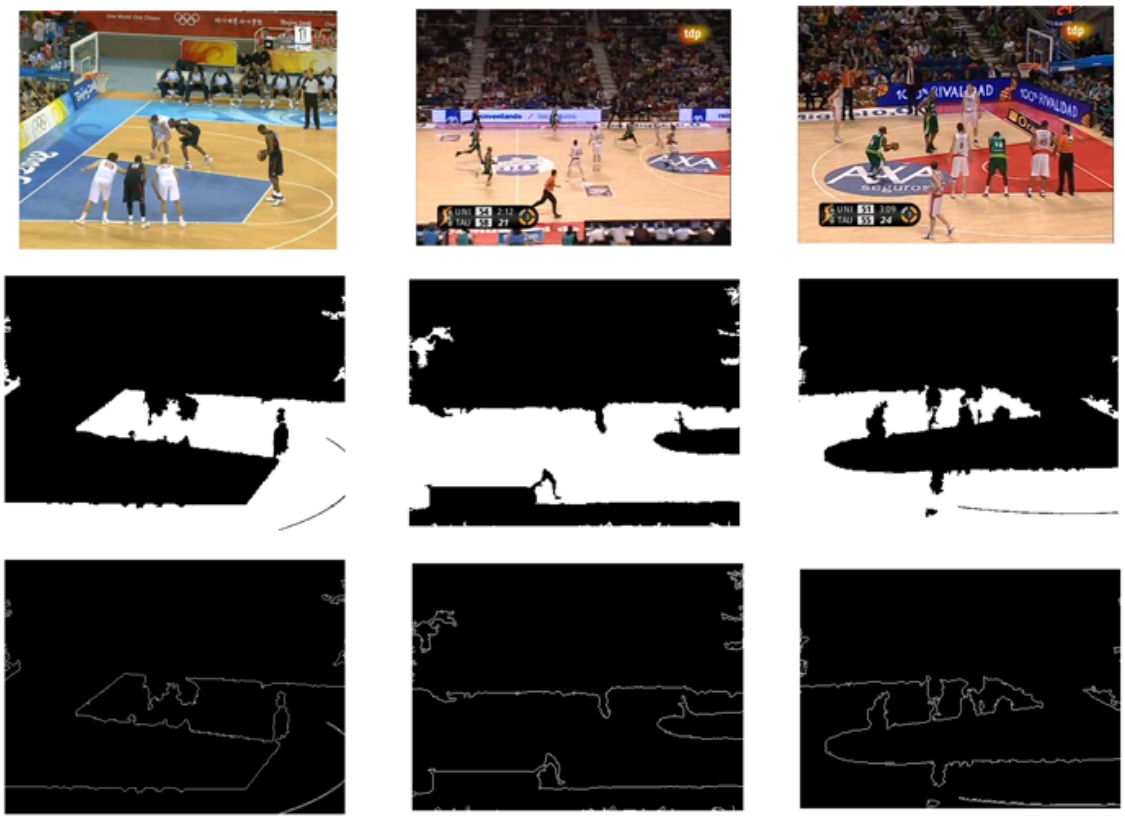

Figura 3.15: Tres ejemplos de la aplicación de detección de bordes a máscaras de campos de baloncesto.

línea de campo, debido a carteles de publicidad, que pueden provocar (del mismo modo que se describe al final de la sección 3.3.4) una delimitación errónea del campo, en este caso al entender que el borde lateral del campo es más exterior de lo que en realidad es.

En todo caso es una técnica que funciona con prácticamente todos los fotogramas del banco de desarrollo y si se compara con las aproximaciones encontradas en la bibliografía, particularmente con las técnicas utilizadas por Chang en [3] y Farin en [11], parece resultar más adecuada que las mismas. En el artículo [3] simplemente se queda con el primer píxel blanco en horizontal de la máscara, los cuales serán en borde real del campo en muy pocos casos y en [11] se detecta la línea blanca en campos de tenis, pero en baloncesto esta línea es extremadamente difícil de detectar.

3.3.6. La transformada de Hough y su aplicación a la detección de líneas

Una vez que hemos extraído una imagen de los bordes de la máscara tratamos de detectar líneas en ellos para así ser capaces de delimitar la frontera del campo. Para detectar estas líneas utilizamos la transformada de Hough.

La transformada de Hough, tal y como fue definida por Richard Duda y Peter Hart en 1972 [6], es generalmente conocida como transformada de Hough estándar (SHT) o transformada de Hough en forma normal (debido al espacio de parametrización que usa).

El concepto en el que se basa es muy sencillo. El conjunto de líneas rectas en un imagen constituyen una familia de dos parámetros. Si fijamos la parametrización para una familia de rectas, entonces cualquier recta arbitraria puede ser representada como un punto en ese espacio. La transformada estándar de Hough usa una parametrización normal, en la que cada recta es representada por los parámetros ρ-θ de la siguiente forma:

$$\rho = x \cdot cos(\theta) + y \cdot sen(\theta) \tag{3.3}$$

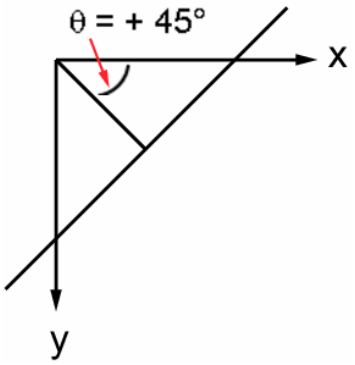

Figura 3.16: Parametrización normal de la recta usada en la SHT (figura extraída de la documentación de la función *hough* de Matlab).

Como se puede ver en la figura 3.16 la variable ρ es la distancia en píxeles desde el origen (esquina superior izquierda de la imagen) a la recta a lo largo de un vector perpendicular a la misma. θ es el ángulo (en grados) de la proyección perpendicular desde el origen hasta la línea desde el eje x positivo en sentido horario. El rango de θ es $-90° \leq \theta \leq 90°$. El ángulo de la recta con el eje x positivo (también medido en sentido horario) será $\theta + 90°$.

Por lo tanto la transformada de Hough es una matriz de un espacio de parámetros cuyas filas y

columnas corresponden a los valores ρ y θ respectivamente. Los elementos de la SHT representan celdas de acumulación. Inicialmente, el valor en cada celda es cero y por cada punto no de fondo en la imagen ($\neq 0$, si es una imagen de bordes como en nuestro caso cada píxel del borde), se calcula el valor de ρ para cada θ y se acumula 1 en el elemento con esa ρ y θ. Al final del proceso un valor Q en una celda (ρ_i, θ_i) representa que existen Q píxeles en el plano XY que caen en la recta definida por (ρ_i, θ_i). Los valores máximos en la SHT representarán potencialmente líneas en la imagen de entrada.

En la figura 3.17 se presenta un ejemplo extraído de la documentación de Matlab (sección *Detecting Lines Using the Hough Transform*) en el que se muestra una representación de la transformada de Hough usando un mapa de colores cálidos. En ella se aprecian algunas propiedades interesantes de esta transformación. Cada punto en la imagen (espacio $x-y$) se corresponde con una curva sinusoidal en el espacio $\rho-\theta$. Por lo tanto puntos de $x-y$ sobre la misma recta corresponden a curvas que pasan por el mismo punto en $\rho-\theta$ y respectivamente, puntos en la misma curva en $\rho-\theta$ corresponden a líneas que pasan por los mismos puntos en $x-y$ [6].

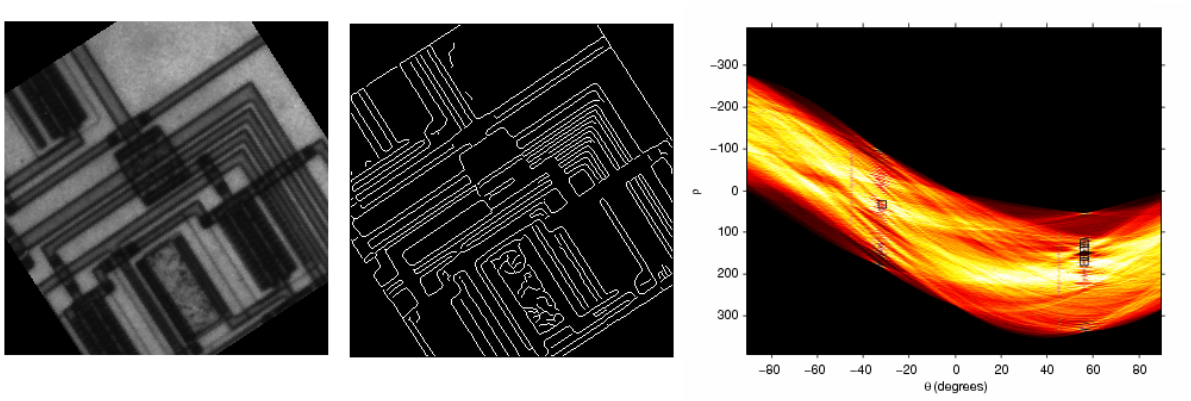

Figura 3.17: Ejemplo de la documentación de Matlab de la utilización de la transformada de Hough para la detección de líneas. Además se presenta el resultado de la función de detección de picos *houghpeaks*.

Si aplicamos esta herramienta sobre nuestra imagen de bordes de la máscara extraída a partir del color del campo y buscamos picos en la matriz resultante, obtendremos un conjunto de rectas entre las cuales con una muy alta probabilidad estarán nuestras líneas de campo.

La transformada de Hough está implementada en Matlab en la función *hough* del *Image*

Processing Toolbox. La función *hough* recibe como parámetro la imagen a la que queremos aplicar la transformada. Además se puede la resolución de ρ que queremos en la transformada mediante el parámetro *'RhoResolution'* con rango $[0, \|tamaño(H)\|]$ (por defecto 1) para especificar el espaciado en ρ de los celdas de la matriz, y mediante *'Theta'* podemos cambiar el rango de θ para el que queremos calcular la transformada.

Para encontrar los picos en la matriz de la transformada de Hough y posteriormente extraer a partir de ellos segmentos utilizamos las funciones auxiliares *houghpeaks* y *houghlines* respectivamente también presentes en el *Image Processing Toolbox*. En lugar de *houghpeaks* se podría utilizar cualquier otra función que detectara picos dentro de la matriz de datos resultante de aplicar la transformada de Hough a la imagen. La utilidad de *houghlines* por otro lado es a partir de esos picos extraer segmentos y como se verá más adelante se puede sustituir por otro tipo de función que realice una tarea similar.

La función *houghpeaks* recibe como parámetros la matriz de la transformada de Hough y el número máximo de picos que deseamos encontrar dentro de la matriz y devuelve la localización de dichos picos (ρ y θ). Adicionalmente se pueden definir otros dos parámetros como son el umbral que utiliza para considerar que un valor de H es un pico (por defecto considera pico aquellos valores que están por encima de $0{,}5 \cdot max(H)$) y una ventana de supresión alrededor del pico en torno a la cual no se buscan otros picos (por defecto el menor valor impar mayor o igual que el tamaño de H entre 50), para que la función no devuelva muchos picos que en realidad son el mismo. En la figura 3.17 se representa con unos marcadores cuadrados negros los picos detectados en la transformada de Hough, especificando un máximo de 5 picos y un umbral de $\lfloor 0{,}3 \cdot max(H) \rfloor$.

La función *houghlines* es algo más complicada. A partir de los picos detectados en la matriz de la transformada de Hough extrae segmentos asociados a dichos picos. Para ello recibe la imagen a la que hemos aplicado la transformada de Hough, los vectores de de rango ρ y θ sobre los cuales se aplica la transformada y los picos detectados. La función nos devuelve los segmentos encontrados en una estructura que tiene como tamaño el número de dichos segmentos en la que se especifican los puntos de origen y final de cada segmento. La versatilidad de la función radica en que es capaz de fusionar segmentos cercanos. Esta fusión se controla mediante los parámetros *'FillGap'*, en el que se define la mínima distancia a la que pueden estar dos segmentos para ser considerados diferentes y que por defecto es 20 píxeles, y *'MinLength'* en la que se define el mínimo tamaño que tiene que tener un segmento, después de la fusión, para no ser descartado, por defecto las

líneas menores a 40 píxeles son descartadas. En la figura 3.18 mostramos el resultado de aplicar *houghlines* al ejemplo que se muestra en 3.17. Los parámetros de *houghlines* en este ejemplo son *'FillGap'=5* y *'MinLength'=7*.

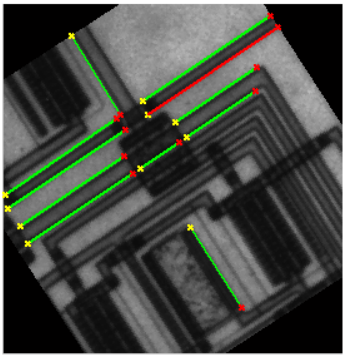

Figura 3.18: Ejemplo de *houghlines* extraído de la documentación de Matlab.

En la imagen se observa que la función *houghlines* devuelve segmentos pertenecientes a cinco líneas distintas ya que ese era el valor de picos detectados en el ejemplo. Además se muestra en rojo, para destacarlo especialmente, el segmento más largo. Como se intuye en este ejemplo el resultado de esta función varía enormemente con el valor de los dos parámetros opcionales de la misma, *'FillGap'* y *'MinLength'*, que deberán ser ajustados para cada problema en particular. Esto como se verá constituye un problema al intentar garantizar cierta robustez en el sistema.

En nuestro problema básicamente tenemos que encontrar líneas pertenecientes a dos familias: las líneas del borde superior del campo y las líneas del borde lateral. Por lo tanto como se ve en el diagrama 3.2 hemos dividido el proceso de detección de líneas en dos etapas. En primer lugar se intentará detectar el borde superior del campo y posteriormente buscaremos el borde lateral del campo.

Esta aproximación tiene varias ventajas: en primer lugar nos fijamos en zonas concretas de la transformada de Hough ya que sabemos que rango de pendientes tienen las rectas (por ejemplo la línea superior de campo nunca será vertical, $\theta \approx 0°$ o $\theta \approx 180°$; en segundo lugar podemos aprovechar la información que extraemos de la línea superior del campo para delimitar la zona en la que encontrar el borde lateral, sabiendo la pendiente del borde superior podemos inferir en que lado del campo nos hayamos, y por último podemos solventar uno de los problemas detectados en la generación y rellenado de la máscara descrita en la sección 3.3.4 (e ilustrado por el fotograma de la derecha de la figura 3.13).

Este problema consistía en que parte del campo era considerada como no de campo al rellenar los huecos "blancos" de la máscara, debido a la situación desafortunada de los jugadores, carteles y otros elementos. Al atacar el problema en dos etapas, podemos utilizar máscaras con diferente rellenado y aprovechar la segmentación parcial del borde superior para solventar este problema. En la sección de detección del borde lateral (sección 3.3.8) se explicará en detalle como llevarlo a cabo.

3.3.7. Detección del borde superior del campo

Como primer paso en la detección de los bordes de campo se tratará de llevar a cabo una detección del borde superior del mismo. Para ello se utilizarán imágenes de bordes extraídas de la máscaras generadas a partir del color del campo como las ya mostradas en la figura 3.15. Como se ve en las mismas el borde superior es perfectamente visible y parece factible que tenga una respuesta dominante en la transformada de Hough. Si aplicamos la transformada de Hough a algunos de los fotogramas del banco de desarrollo y posteriormente buscamos picos en la misma con la función *houghpeaks* (un máximo de 10 picos) obtenemos los siguientes resultados (figura 3.19).

Como se ve en la figura, se obtienen siempre los picos para valores de θ cercanos a 90° o -90° (bordes de la representación de la transformada). Esto parece correcto ya que el borde superior, también el inferior aunque no estamos interesados en él, tienen una θ cercana a ese valor (ver figura 3.16 en la que se ve como se define este ángulo). Por otro lado también se observa, aunque no claramente en la figura, que no siempre obtenemos 10 picos como resultado de aplicar *houghpeaks* sino que sólo se devuelven aquellos que superen el umbral (en el caso del tercer fotograma sólo dos).

El número de picos a detectar fue elegido de manera experimental, así como el valor del umbral (que finalmente se fijó a su valor por defecto $0{,}5 \cdot max(H)$). Es importante que dicho valor sea suficientemente holgado como para que bordes no muy claros (como en el segundo caso) sean detectados y suficientemente estrecho como para que no se detecten muchos falsos picos que puedan reducir el rendimiento del sistema.

Una vez que se obtienen los picos, tratamos de construir líneas a partir de ellos del mismo modo que se hace en el ejemplo presentado en 3.3.6. En este caso, como se ha dicho, el valor que se seleccione para los parámetros de la función *houghlines* resultan decisivos para el correcto

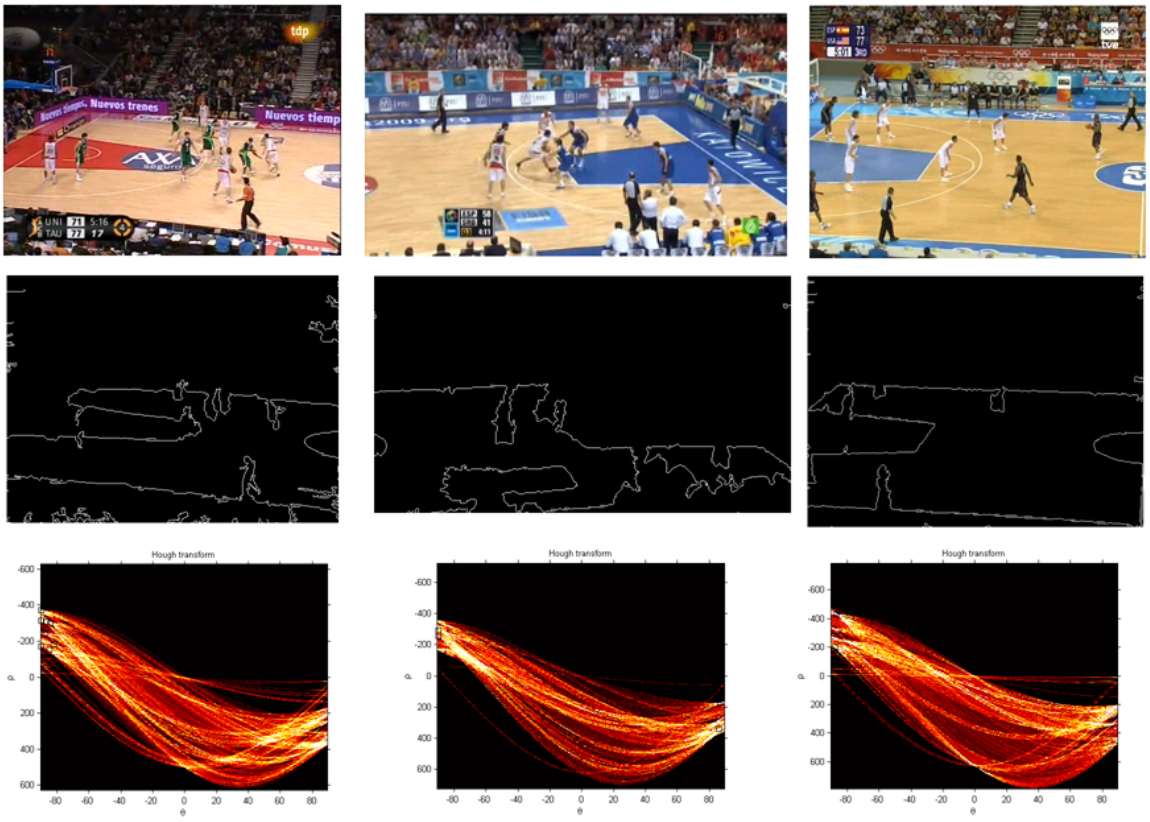

Figura 3.19: Transformada de Hough de los bordes de varios fotogramas de ejemplo.

funcionamiento de la misma. En nuestro caso tenemos que construir líneas largas que en número de píxeles miden como mínimo cerca de mitad de la imagen. Por ello los valores tanto de *FillGap* como de *MinLength* tienen que ser altos. Después de numerosas pruebas dichos valores fueron ajustados a 100 y 60 píxeles respectivamente. Hay que recordar que las resoluciones de los vídeos, es decir el número de píxeles de los fotogramas en horizontal y vertical, con los que estamos trabajando están descritas en las tablas 3.1 y 3.2. En la figura 3.20 se ve el resultado de aplicar *houghlines* a los fotogramas del ejemplo anterior.

En la figura se muestran en azul el segmento más largo y en verde el resto. Como se ve en todos los casos se obtiene un conjunto de segmentos candidatos a borde superior entre los que se encuentra el correcto. El problema en este momento radica en saber identificar este segmento. Ya que el criterio trivial de seleccionar el segmento más largo o el de respuesta más dominante no es valido, habrá que encontrar un criterio que nos permita identificar el borde correcto. Hay que destacar que en algunos casos (como por ejemplo en el primer fotograma del ejemplo) existen

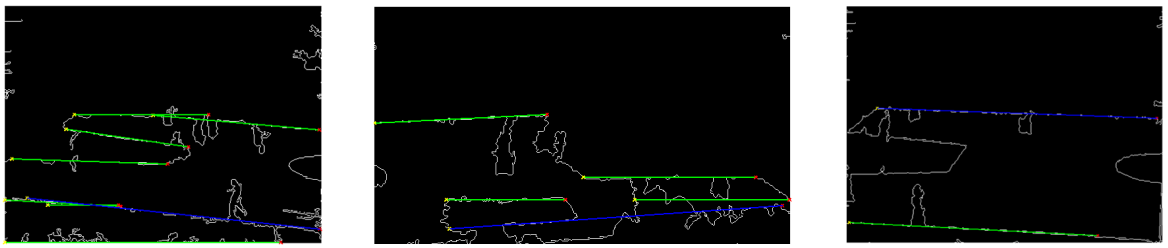

Figura 3.20: Resultado de detectar segmentos con *houghlines* en el ejemplo anterior.

dos posibles bordes casi equivalentes que difieren ligeramente en la pendiente.

En este punto nuestra aproximación asume dos hipótesis. En primer lugar que la máscara extraída constituye una buena representación del campo, si eso es así y el borde superior se encuentra entre los candidatos obtenidos al aplicar la transformada de Hough (segunda hipótesis) al segmentar la imagen en dos verticalmente utilizando como frontera los candidatos a borde superior habrá un número reducido de píxeles de campo (píxeles blancos) en el segmento superior cuando el candidato sea un buen borde superior.

Por lo tanto la forma de operar es la siguiente. Se construye un polígono a partir de la extensión de cada segmento obtenido hasta cortar con los límites de la imagen. Con este polígono se construye una máscara que se comparara con la máscara que tenemos extraída (tal y como se explico en la sección 3.3.4) y se compararan el número de errores contabilizados (píxeles blancos en la parte superior) con un umbral que dependerá del área de la máscara original (número total de píxeles blancos de la máscara). Si este número de errores es menor que el umbral el candidato será considerado como el borde superior correcto. Todo el procedimiento está ilustrado en la figura 3.21.

Para construir el polígono que segmenta la imagen verticalmente hay que extender los segmentos hasta cortar el límite de la imagen. Esto se hace mediante las siguientes ecuaciones donde (x_a, y_a) y (x_b, y_b) son los puntos iniciales y finales que definen los segmentos y x_{max} e y_{max} son el ancho y el alto de la imagen respectivamente. En la última de las ecuaciones se muestras los puntos del polígono, numerados desde la esquina superior derecha en sentido antihorario.

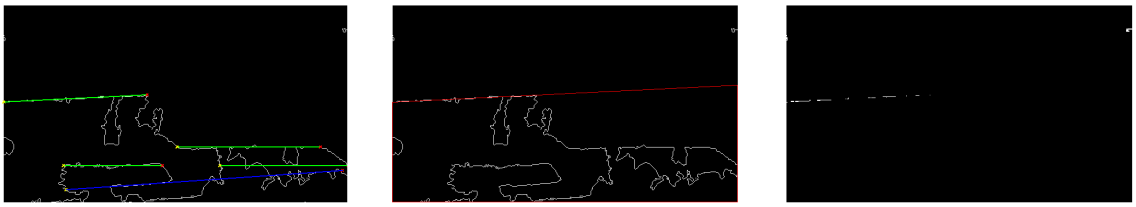

Figura 3.21: Procedimiento de selección de candidatos a borde superior. En primer lugar se construye un polígono con cada candidato y posteriormente se extraen el número de píxeles de campo que deja por encima de la frontera.

$$\begin{gathered} m = \frac{y_b - y_a}{x_b - x_a} \\ x_0 = x_a \\ y_0 = y_a \\ y = m \cdot (x - x_0) + y_0 \\ Puntos: \ (1, y(1)); (1, y_{max}); (x_{max}, y_{max}); (x_{max}, y(x_{max})) \end{gathered} \tag{3.4}$$

A partir de este polígono se puede construir una máscara del candidato que se puede comparar con la máscara de color para ver que píxeles de error de campo se obtendrían. Para compararla con la máscara de color simplemente llevamos a cabo una operación AND de la máscara de color con el inverso de la máscara del candidato. En la figura 3.22 se ve un ejemplo de esta operación.

Una vez que tenemos la máscara de los píxeles de campo de error se suman y esta suma la comparamos con un umbral. El umbral es fijado experimentalmente al siguiente valor.

$$errores_{pixeles_{campo}} < 0{,}2 \cdot total_{pixeles_{campo}} \tag{3.5}$$

donde $total_{pixeles_{campo}}$ es la suma de todos los píxeles de campo de la máscara de color.

La máscara de candidato se puede generar fácilmente con la función *poly2mask* y las dos sumas de píxeles anteriores se pueden realizar con la función *bwarea*, ambas funciones del *ImageProcessing Toolbox*.

En la figura 3.23 se observa el resultado final del método de detección de borde superior aplicado a los tres fotogramas anteriores. Como se verá en la sección de análisis de resultados 3.3.9, este método tiene una alta tasa de acierto aunque sus fallos y limitaciones serán presentados

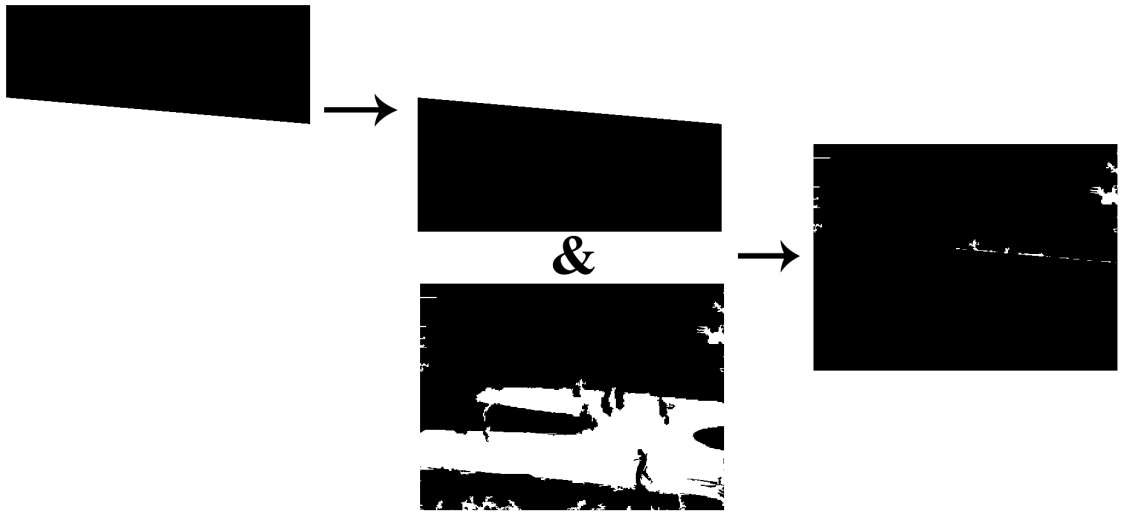

Figura 3.22: Procedimiento de cálculo de errores de segmentación vertical. La máscara del borde candidato se niega y se multiplica por la máscara de color, el resultado son los errores de tipo 1 en esa sección.

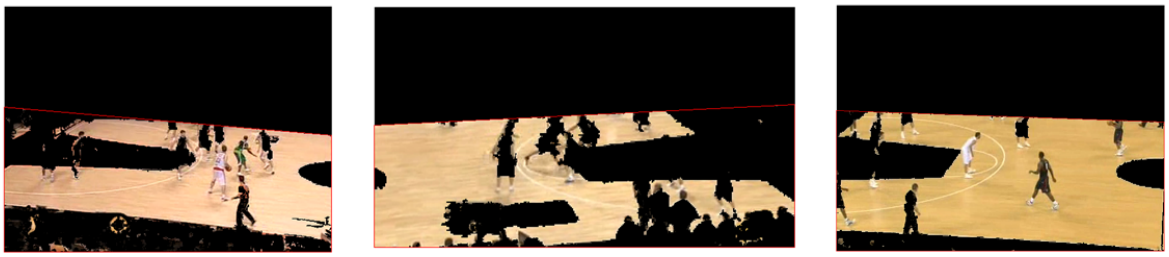

Figura 3.23: Ejemplo de resultados de bloque de detección de borde lateral

Una vez detectado este borde superior utilizaremos la máscara que generamos con el mismo para al multiplicarla máscara de color y eliminar todos los errores existentes por encima del borde superior. Antes de hacer esto dilatamos la máscara verticalmente, es esta forma el borde superior queda un poco por encima de la línea detectada, evitando que la segmentación sea demasiado estricta. Esta operación se puede realizar en Matlab mediante la función *imdilate* del *Image Processing Toolbox*.

En vez de utilizar la misma máscara de color que hemos usado para esta detección, podemos

usar la máscara en la que sólo han sido rellenos los huecos de fondo (negros), es decir la que surge del primer rellenado, evitando así el problema de que al rellenar los huecos blancos, perdamos parte del campo. Multiplicando por tanto la máscara de borde superior por esta máscara de color, obtenemos una nueva máscara que no tiene errores de tipo 1 por encima del borde superior elegido. Esta máscara está lista para ser la entrada del siguiente bloque de detección del borde lateral.

Este borde lateral será el borde izquierdo en algunos casos, en otros el derecho y en otros el plano será del centro del campo y por lo tanto no se verá en la pantalla. El lado del campo en el que estamos está directamente relacionado con la pendiente del borde superior, ya que la cámara está situada en el centro del campo. Por lo tanto cuando la pendiente del borde superior sea positiva estaremos en el campo izquierdo y cuando esta sea negativa estaremos en el campo derecho. Fijamos un umbral a partir del cual consideramos factible la existencia de una línea lateral en la imagen. Este umbral lo fijamos experimentalmente a:

$$umbral_{lado} = 0{,}04 \tag{3.6}$$

Si la pendiente es mayor que $umbral_{lado}$ buscaremos un borde lateral izquierdo, si es menor que $-umbral_{lado}$ buscaremos un borde lateral derecho y si está entre estos dos valores el siguiente bloque no hará nada, ya que la presencia de un borde lateral no es factible.

3.3.8. Detección del borde lateral del campo

Este bloque se basa en el mismo principio que el bloque anterior. Como se ha dicho, como entrada tiene la máscara extraída a partir de la máscara de color y la segmentación de borde superior, el borde superior (en forma de polígono de 4 vértices) y la pendiente del mismo.

Ya se ha comentado que la pendiente del borde superior nos da información de en que lado del campo nos encontramos. Lo cierto es que esto es una información de alto nivel de cierta importancia, y probablemente de las pocas que puede extraer este sistema por si solo. El saber en que campo nos encontramos nos puede proporcionar información de que equipo está atacando que nos puede permitir extraer estadísticas o utilizarlo junto con otra información para detectar eventos. Por lo tanto entre las medidas de exactitud del sistema tendremos también en cuenta la capacidad de mismo para detectar correctamente el lado del campo en él que nos encontramos.

Pero en este punto la utilidad de saber en que lado del campo nos encontramos radica en

que nos permite buscar bordes sólo en un rango de posibles direcciones. Esto se concreta en que buscamos picos de la transformada de Hough no en todo el espacio de parámetros sino para ciertos rangos de θ y ρ. Esto es importante ya que como se aprecia en las imágenes (por ejemplo de la figura 3.19), los bordes laterales son mucho menos claros por lo que la respuesta de la transformada de Hough no es tan importante y es más difícil encontrar picos. Por ello fijamos el rango de θ a los siguientes valores (tabla 3.3) para los dos casos. El rango de ρ no lo fijamos en esta versión ya que es difícil encontrar un rango funcional para todos los diferentes vídeos en los que difiera, ligera pero apreciablemente, la altura y la distancia de la cámara.

	θ
Borde Izquierdo	(20°,70°)
Borde Derecho	(-70°,-20°)

Tabla 3.3: Rango de valores de θ para ambos casos de detección de bordes laterales

Además se amplia el número de picos a buscar a 30 ya que los picos ahora son menos destacados y se reduce el mínimo tamaño de segmento a 20 píxeles ya que ahora los bordes serán más pequeños que el borde lateral. El resultado obtenido por estas funciones está ilustrado en la figura 3.24.

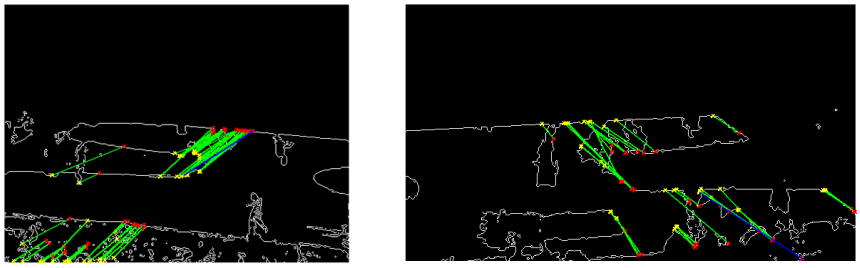

Figura 3.24: Ejemplo de aplicación de *houghlines* a la detección de bordes laterales

Como se ve en la figura, que sigue el mismo ejemplo anterior pero donde uno de los fotogramas se considera de centro de campo por lo que no se trata detecta el borde lateral, el sistema funciona bastante peor a la hora de detectar las rectas que definen los bordes laterales. Como e ve se detectan muchas líneas falsas y en algún caso entre los candidatos ni quiera está el borde correcto (debido a que el borde es bastante rugoso).

Si consideramos que entre los candidatos si está el borde correcto, para segmentar el campo

procedemos de mismo modo que en caso anterior: a partir de los segmentos candidatos a borde construimos polígonos y elegimos el que mejor segmente el campo. Al polígono de cuatro puntos de la segmentación anterior, tenemos que añadir un punto más que se corresponda a la intersección entre los dos bordes y modificar el primer (o cuarto en el caso del borde derecho) punto para que pertenezca al borde lateral. Este proceso está ilustrado en la figura 3.25, en ella se muestra en rojo el polígono generado por el bloque de detección de borde superior anterior y en verde el que se generaría en este bloque según consideremos que estamos en el lado izquierdo o derecho.

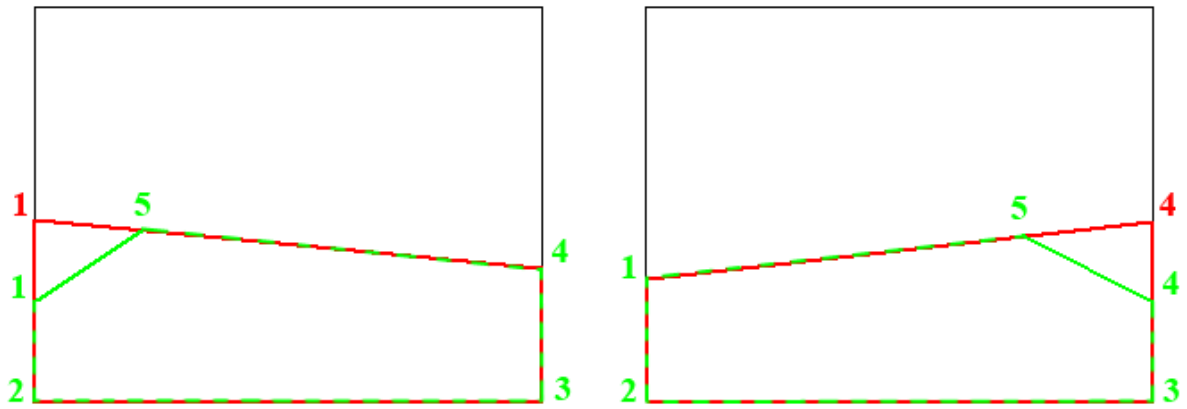

Figura 3.25: Procedimiento de delimitación del campo para lado izquierdo y derecho. A partir de la segmentación de borde superior (línea roja) y dependiendo del lado en que nos encontremos generaremos un nuevo polígono frontera incluyendo el borde lateral candidato (línea verde).

La ecuación de este punto intersección (punto 5 del borde) es la que soluciona la que sale de intersecar ambos bordes, es trivial demostrar que se valor es :

$$\begin{aligned} m_2 &= \frac{y_b - y_a}{x_b - x_a} \\ x_{0_2} &= x_a \\ y_{0_2} &= y_a \\ x_5 &= \frac{y_{0_2} - y_{0_1} + m_1 \cdot x_{0_1} - m_2 \cdot x_{0_2}}{m_1 - m_2} \\ y_5 &= m_2 \cdot (x_5 - x_{0_2}) + y_{0_2} \end{aligned} \tag{3.7}$$

donde (x_a, y_a) y (x_b, y_b) son los puntos iniciales y finales que definen los segmentos de borde lateral y m_1 es la pendiente del borde superior, por lo tanto el polígono que surge de esta

segmentación estará descrito de la siguiente forma ($y_{4_{up}}$ e $y_{1_{up}}$ son los puntos descritos por la ecuación del borde superior):

$$y_1 = \begin{cases} m_2 \cdot (1 - x_{0_2}) + y_{0_2} & \text{si borde lateral derecho} \\ y_{1_{up}} & \text{si borde lateral izquierdo} \end{cases}$$

$$y_4 = \begin{cases} y_{4_{up}} & \text{si borde lateral derecho} \\ m_2 \cdot (x_{max} - x_{0_2}) + y_{0_2} & \text{si borde lateral izquierdo} \end{cases} \tag{3.8}$$

$$Puntos: \ (1, y_1); (1, y_{max}); (x_{max}, y_{max}); (x_{max}, y_4); (x_5, y_5)$$

El umbral de error que se utiliza para selección del mejor de los candidatos se calcula de igual forma que en el caso anterior aunque se reduce el porcentaje de errores de campo visibles al 10 %.

$$errores_{pixeles_{campo}} < 0{,}1 \cdot total_{pixeles_{campo}} \tag{3.9}$$

En la figura siguiente 3.26 se puede ver un ejemplo del resultado del sistema completo. La máscara final es también dilatada sencillamente a fin de que se vea bien el resultado de la segmentación.

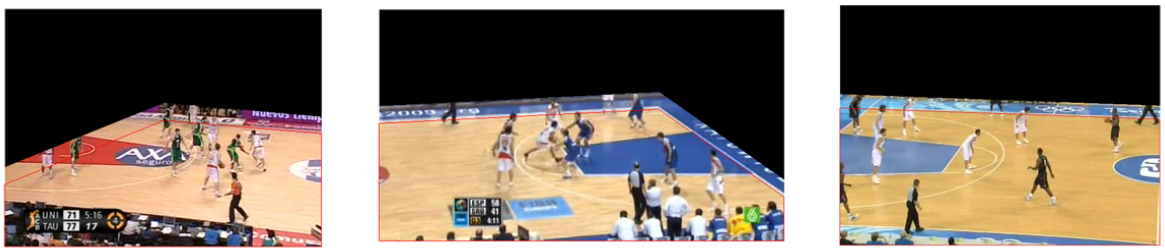

Figura 3.26: Ejemplo del resultado de la segmentación

En la sección siguiente se analiza críticamente el resultado de aplicar el sistema descrito al banco de pruebas de desarrollo, para identificar problemas de diseño y posibles mejoras que se aplican en la versión siguiente.

3.3.9. Análisis de resultados

Si aplicamos al sistema descrito en las secciones anteriores el banco de prueba de desarrollo y valoramos como se describió en la sección 3.2 la tasa de acierto en la detección del borde superior, la tasa de acierto en la detección del lado de campo en la que nos encontramos y la tasa de acierto en la detección del borde lateral obtenemos los siguientes resultados:

Resultados de la versión inicial 0.1			
	Detección de borde superior	**Clasificación de lado de campo**	**Detección de borde lateral**
Test 1	84 %	72 %	44 %
Test 2	64 %	62 %	56 %

Tabla 3.4: Resultados del banco de pruebas de desarrollo al aplicarlo en la versión 0.1

En el primer test se observa que se obtiene una alta tasa de acierto en la detección del borde superior y en la clasificación del lado del campo mientras que en la detección del borde lateral el resultado es bastante malo. Algo parecido ocurre en el caso del test 2 pero en ese caso la segmentación del borde superior es notablemente peor.

En el test 2 se utilizan 5 secuencias de 5 vídeos distintos y los errores normalmente vienen por ráfagas, así en aquellos casos en los que un fotograma falle es muy probable que falle más de uno ya que no se diferencian mucho (entre un fotograma y el siguiente hay 40 milisegundos). Lo que ocurre básicamente es que existen dos secuencias que presentan la mayor parte de los fallos mientras en las otras tres los resultados son buenos. En el test 1 al ser fotogramas de momentos separados una mala clasificación en un fotograma no implica errores en otros, aunque si es cierto que existen vídeos en los que los errores son mayores. En este sentido la tendencia es que los partidos en los que el borde del campo es rojo (matiz más cercano al marrón que buscamos) tengan más errores en la detección de borde lateral.

Los errores en la detección de borde superior son más graves ya que implican que por lo general no se pueda detectar el borde lateral (ya que se utiliza la segmentación anterior como base) y estos parecen no estar correlados con el partido al que pertenezca el vídeo sino con la secuencia concreta y el efecto que la situación de los jugadores tiene en la máscara.

En la siguiente figura 3.27 se ve algún ejemplo de los errores típicos de segmentación detectados en esta versión, junto con una imagen de los bordes y las líneas generadas por a partir de

los picos detectados en la transformada de Hough.

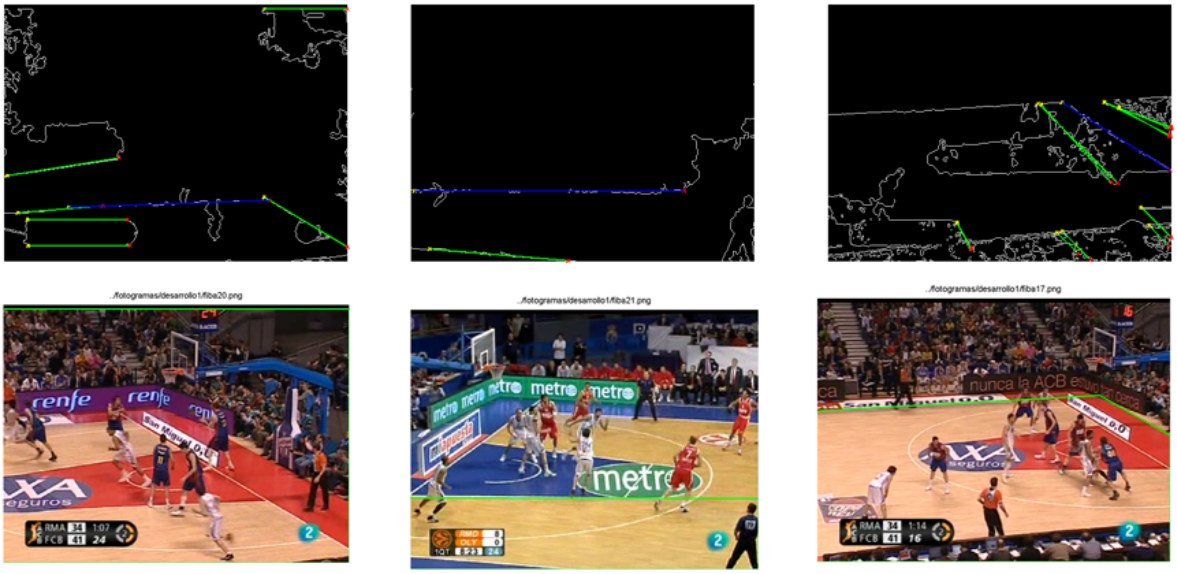

Figura 3.27: Algunos ejemplos de errores típicos encontrados en la versión 0.1

En el primer fotograma se ilustra el problema derivado de que haya grandes zonas que se detecten como de campo en la parte del público y que no se eliminen en el rellenado de la máscara y que tengan una respuesta que siga aproximadamente una recta. Esto es debido a dos cosas, en primer lugar al método usado para generar la máscara de color y en segundo lugar a que extraemos tanto los bordes que van de negro a blanco como de blanco a negro, cuando en el caso que nos ocupa (borde superior) sólo deberíamos extraer bordes de campo a no campo, es decir de blanco para negro. También aunque esta segmentación es muy mala, debido a que no contabilizamos los errores de tipo 2 (píxeles que no son de campo en la zona de campo) ya que son potencialmente mucho más que los de tipo 1 y provocarían segmentaciones erróneas, esta segmentación es considerada muy buena por el sistema ya que no tiene errores de tipo 1 (no hay nada de campo por encima de la línea). Además es obvio que la línea superior de campo, que puede estar en diferentes posiciones de la imagen dependiendo de la toma, nunca estará tan arriba.

El fotograma 2 ilustra un error más complejo ya que el borde es muy poco claro debido al rellenado de la máscara, este error tiene difícil solución y abordaremos su solución en la segunda mejora 3.5 cuando añadamos procesado temporal al sistema.

En el tercer fotograma tenemos un error en el borde lateral debido a que hay una zona de parquet paralela al campo fuera de él. Este error es similar al error del primer fotograma en el sentido de que también un borde de no campo a campo es detectado y considerado entre los candidatos. También hay que notar que la función *houghlines* no aporta nada a excepción de la proyección del sistema de referencia de la transformada de Hough al de la imagen, y añade una complejidad al sistema dado el difícil ajuste de sus parámetros, por lo que una función sustitutiva de esta, probablemente mejorara el sistema, como se ve en la sección 3.4.

Para solucionar todos estos problemas se implementaron las siguientes mejoras en la versión 0.2 (sección 3.4):

1. Mejoras menores en la detección de color para que los bordes sean más identificables.
2. Modificación de la función de extracción de bordes por una que tenga en cuenta la orientación del borde.
3. Delimitar más los rangos de la transformada de *hough* en los que se buscan picos para no fijarnos en rectas que en ningún caso delimitarán el borde.
4. Reemplazo de la función *houghlines* por una más sencilla, adaptada al problema y fácil de ajustar.
5. Mejora del criterio de selección del mejor candidato contemplando tanto los errores de tipo 1 como los errores de tipo 2, debidamente pesados.

3.4. Primera Mejora. Versión 0.2

En esta sección se describen las mejoras que se implementaron en el sistema a partir de las limitaciones detectadas en la versión anterior. Básicamente estas mejoras consisten, además de ciertos ajustes en algunos parámetros, en la implementación de una nueva función para extraer los bordes de la máscara de campo, de una nueva función que extrae a partir de estos las rectas candidatas a ser la frontera del campo y de un nuevo criterio de selección de candidatos que tenga en cuenta de manera adecuada los dos tipos de errores que presentan las segmentaciones.

Dado que la estructura del sistema es la misma que la anterior, se presentaran las mejoras en el mismo orden que la sección anterior presentando únicamente las subsecciones necesarias.

3.4.1. Detección del color de campo

Como se ve en la sección 3.3.3 este bloque funciona bastante bien a la hora de encontrar la zona que presenta el color de campo, aunque se puede ver que en algunos casos patológicos, especialmente cuando el exterior del campo está pintado de rojo (H en torno a 0 o 1), la máscara que se extrae tiene unos bordes muy rugosos lo que provoca que tengan una respuesta menos clara en la transformada de Hough. Por ello en esta versión se ajustaron ligeramente los valores de los parámetros del rango de matiz (R_h1) y tolerancia ($Tolerance_H$) usados en 3.3.3 a unos valores que dieran un mejor resultado en base a la pruebas realizadas. Finalmente se seleccionaron los siguientes valores:

$$\begin{aligned} R_h1 &= [0{,}04\ ,\ 0{,}15] \\ Tolerance_H &= 0{,}02 \end{aligned} \tag{3.10}$$

Además con el fin de hacer que las zonas de saturación muy baja (S de valor cercano a 0), es decir zonas de color blanco independientemente del valor de H, no degradaran la detección, a píxeles con una saturación menor que 0.25 se le asignó un valor de "no campo" (valor 0) a la hora de generar la máscara con independencia del valor de H. Adicionalmente también se asignaron un valor de "no campo" (valor 0) a aquellos píxeles con mucha saturación (mayor que 0.75) ya que el campo tiende a tener un valor de saturación media. En cuanto a la componente V es contraproducente fijar ningún límite debido a que hay que contemplar los diversos brillos y sombras que hay sobre el campo de juego. En la figura 3.28 se ve un ejemplo de lo que se consigue con esta mejora.

Estos dos ajustes menores consiguen mejorar ligeramente las máscara que se genera (de la misma forma que en la versión anterior), logrando que esta tenga unos bordes laterales algo más suaves y por lo tanto que la transformada de Hough presente picos más claros donde antes no los mostraba.

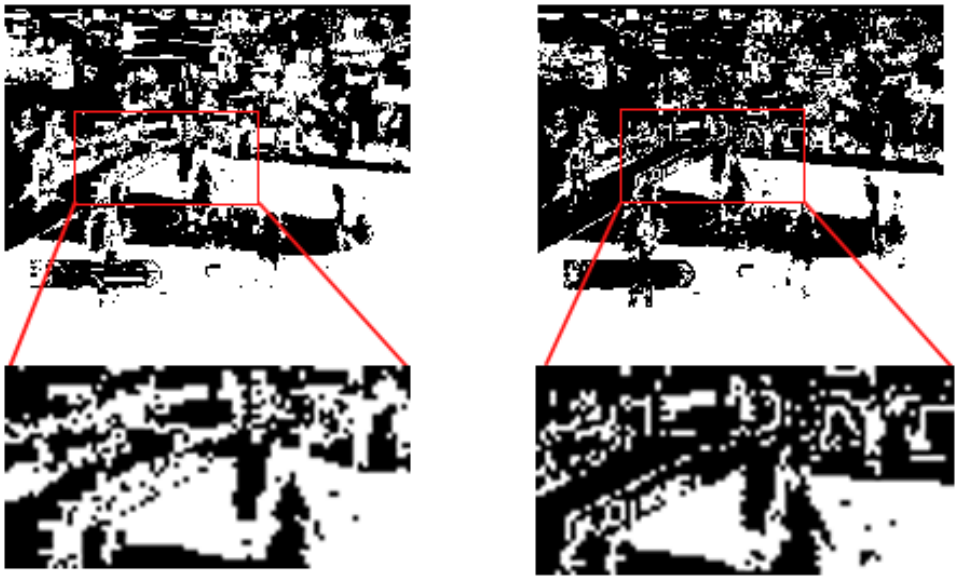

Figura 3.28: Efecto de aplicar la correción de la máscara basada en la saturación. En la columna de la izquierda aparece una máscara en la que no se ha corregido la saturación y un detalle de la misma, en la derecha aparece la misma máscara aplicandole dicha corrección.

3.4.2. Extracción de bordes de la máscara

Como se vio en los resultados de la versión 0.1 (sección 3.3.9), el sistema extrae tanto los bordes de la máscara que van de blanco a negro como los que van de negro a blanco. Esto resulta problemático en algunos campos ya que aparecen falsos bordes en zonas exteriores al campo del mismo color que el mismo (ver tercer fotograma de la figura 3.27). La forma de solucionar esto es utilizando una función de extracción de bordes que sólo los devuelva en el caso de que el cambio sea de blanco a negro en dirección hacia el borde que se desea extraer (de la región a segmentar,

Figura 3.29: Ejemplo de la técnica de extracción de bordes. Se superpone a la máscara una copia de la misma (color amarillo translucido) desplazada hacia arriba. En negro se ven los bordes, que como se ve son los que presentan un cambio de región de campo a región no de campo en sentido ascendente.

el campo de juego, hacia afuera).

Aunque podríamos usar una técnica de filtrado paso alto más compleja (similar a los operadores de Sobel [12] que utiliza *Matlab* en *edges*), debido a que los bordes se extraen de una máscara (que es una imagen binaria sumamente sencilla) y dado que pretendemos, en la medida de lo posible, que el sistema sea eficiente (computacionalmente hablando), optamos por la utilización de la técnica más simple posible para la extracción de bordes: una técnica basada en la primera diferencia.

La operación es tan simple como restar a la máscara una copia de ella misma desplazada un píxel en la dirección apropiada para detectar alguno de los bordes. En la figura 3.29 se ve un ejemplo de la utilización de esta función. Se ve una parte de una máscara de campo, a la que se le ha superpuesto una copia de si misma (pintada de amarillo) desplazada un píxel hacia arriba. Como se ve sólo se extraen los bordes que suponen un cambio de "campo" a "no campo" en dicho sentido, en la figura son las líneas negras.

Además de su obvia sencillez esta técnica tiene la ventaja buscada, sólo nos resalta los bordes en una dirección y sentido concretas, eliminando el efecto que puedan tener otros bordes erróneos en las posteriores operaciones.

Para la detección del borde superior. Desplazamos la máscara hacia arriba una posición y nos quedaremos sólo con aquellos bordes de valor positivo. Por lo tanto la imagen de bordes devolverá como píxeles blancos (valor 1) aquellos bordes que delimiten un cambio de blanco a negro en dirección vertical y sentido ascendente.

Para la detección de los bordes laterales. Desplazaremos la máscara un píxel hacia arriba y un píxel hacia la derecha o la izquierda, dependiendo de que queramos detectar un borde lateral derecho o izquierdo respectivamente. De la misma forma que el caso anterior obtendremos un borde en aquellas transiciones de blanco a negro en dirección 45º o -45º (medidos con respecto a la vertical) en sentido ascendente.

En las siguiente figura 3.30 se ve un ejemplo de la aplicación de esta técnica a algunos fotogramas del banco de pruebas, para la extracción del borde superior.

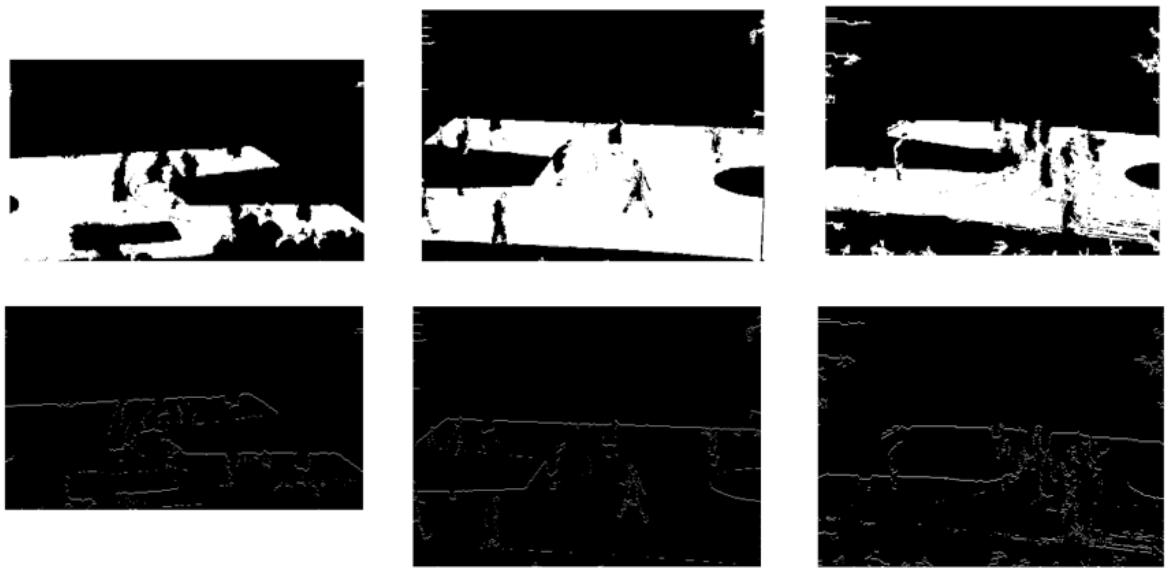

Figura 3.30: Ejemplo de extracción de bordes para detección del borde superior a partir de la máscara de color.

Si se compara con los resultados de la técnica utilizada en la versión 0.1 (ver figura 3.15) se ve que aunque los bordes son menos claros, hemos eliminado gran cantidad de bordes falsos o que no nos interesan y que antes aparecían muy claramente. Por ello podemos pensar que la transformada de Hough tendrá una respuesta proporcionalmente más clara en los bordes correctos que antes.

3.4.3. Mejoras en la detección de bordes

Una vez que hemos mejorado nuestra técnica de extracción de bordes para eliminar respuestas de los que no nos interesan, nos centraremos en mejorar la forma en la que a partir de la

transformada de Hough extraemos candidatos a borde de campo.

La base de la versión anterior sigue siendo válida. A partir de la transformada de Hough de la imagen de bordes extraída en el paso anterior, detectamos los picos (limitándolo a una región concreta para eliminar en la medida de lo posible respuestas falsas). Y a partir de las direcciones que dichos picos definen formamos un polígono que sea frontera del campo de juego, primero sin considerar que existe un borde lateral y posteriormente (si este aparece en la imagen) incluyendo este como uno de los lados del polígono segmentador.

Algunos de los problemas detectados en la versión anterior (ver sección 3.3.9) son debidos a respuestas como bordes claramente fuera de la posición del campo. Aunque la mejora de la sección anterior (3.4.2) contribuye a reducir la importancia de estos borde, se puede llevar a cabo otra mejora como es, sólo buscar bordes a partir de una cierta distancia ρ_{min} del origen de coordenadas (recordar que dicho origen es la esquina superior derecha de la imagen y que ya en la versión anterior llevamos a cabo una delimitación de la zona de la transformada en la que se buscan los picos aunque sólo en la coordenada θ y para el borde lateral). Hay que tener en cuenta que estos límites no pueden ser muy estrictos ya que la posición de los campos en la imagen puede diferir enormemente dependiendo de la posición de la cámara en el campo, por ello fijamos los siguientes:

	ρ
Borde superior	$(\frac{y_{max}}{3},\infty)$
Borde lateral izquierdo	$(0,y_{max})$
Borde lateral derecho	$(0,\infty)$

Tabla 3.5: Rango de valores de ρ para la detección de borde superior y ambos casos de bordes laterales.

El resto de problemas detectados en la versión anterior están principalmente causados por la implementación de Matlab de la función de extracción de líneas a partir de los máximos *houghlines*. Como ya se comentó en la sección 3.3.6 esta función es bastante versátil pero a cambio tiene ciertos parámetros que son difíciles de ajustar para que sean válidos para todos los posibles casos que queremos cubrir, deteriorando la robustez del sistema. Por ello de planteó obtener los posibles bordes directamente a partir de los picos que te devuelve *houghpeaks*, proyectando esos valores al sistema de coordenadas necesario para construir los polígonos de los candidatos a segmentaciones.

Como se comenta en la sección 3.3.6, los picos detectados están definidos por su valor de ρ y θ con respecto al origen de coordenadas. Para extraer las coordenadas (x,y) de los puntos que forman el polígono de la segmentación tenemos que obtener la proyección de las direcciones detectadas encontrando los puntos que corten con los dos extremos de la imagen ($x = 1$ y $x = x_{max}$), en el caso del borde superior, y con el extremo correspondiente y el borde superior en el caso de los bordes laterales. Estos puntos serán los denominados (x_a, y_a) e (x_b, y_b) que se incluirán en las ecuaciones 3.4, 3.7 y 3.8 descritas anteriormente. Esta proyección se hace como:

$$\begin{aligned} x_a &= x(1) \\ x_b &= x_{max} \\ y_a &= \left\lfloor \left| \frac{\rho_{peak}}{\cos(90° - \theta_{peak})} \right| \right\rfloor \\ h &= x_{max} \cdot \tan(90° - \theta_{peak}) \\ y_b &= y_a - h \end{aligned} \tag{3.11}$$

Donde x_{max} es la anchura de la foto, número de píxeles en esa dirección.

Como se ve en la figura 3.31 al aplicar estos nuevos métodos a algunos fotogramas de ejemplo para obtener el borde superior, obtenemos una serie de candidatos (todos ellos en los dos tercios inferiores de la imagen debido al límite impuesto) entre los que se encuentra el borde correcto.

Posteriormente a partir de los candidatos de la imagen habrá que seleccionar el borde correcto, para ello se hará uso del nuevo criterio que se explica en la sección 3.4.4. Una vez que se tenga ese candidato si aplicamos las técnicas aquí descritas a la detección del borde lateral obtendremos los resultados ilustrados en la figura 3.32, donde se muestran los candidatos a borde lateral, junto con la elección final.

Como se ve en este ejemplo casi todos los bordes laterales detectados son muy similares y similares al borde real. Al ser este el mismo ejemplo que el presentado en la versión anterior (sección 3.3.8) se observa en este punto una notable mejora en los resultados. Por otro lado el hecho de que los bordes sean tan similares hace que la elección de uno u otro sea poco importante; pero no en todos los casos será así por lo que debemos diseñar unos buenos criterios de selección que permita seleccionar que bordes, tanto laterales como superiores, son los correctos. Este criterio se presenta en la sección siguiente.

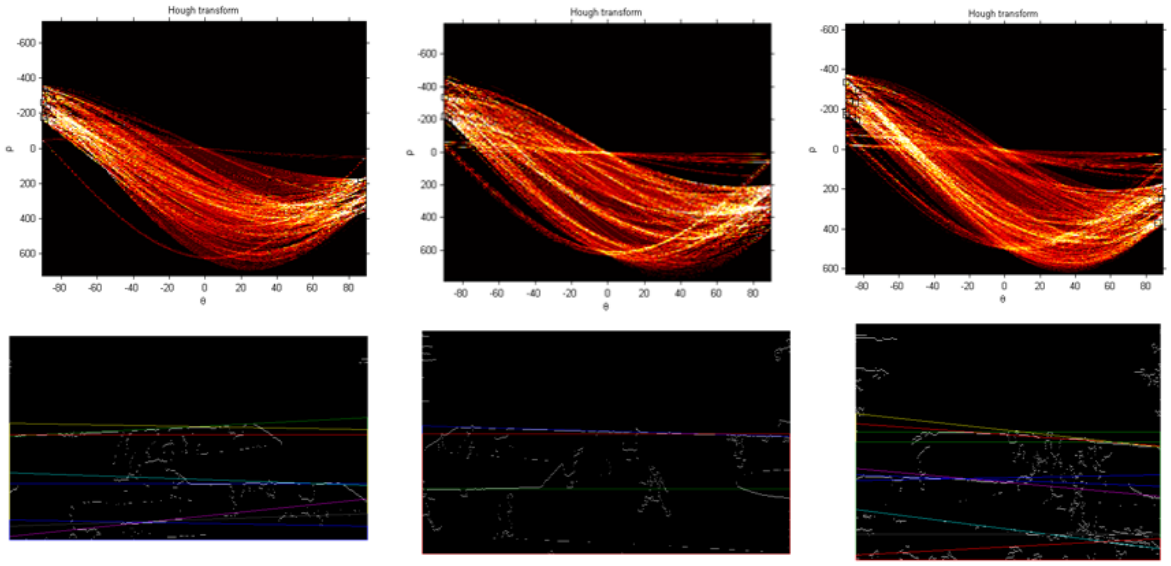

Figura 3.31: Ejemplo de la aplicación de la transformada de Hough a los bordes de tres fotogramas y de la nueva técnica de extracción de líneas para identificar el borde superior.

3.4.4. Mejora en el criterio de selección de candidatos

En la versión anterior se selecciona que borde es el mejor candidato siguiendo un criterio en el que se tienen en cuenta simplemente los errores de tipo 1, píxeles identificados como de campo fuera de la zona segmentada como campo, dando un resultado no del todo satisfactorio. Esto se hace así porque los errores de tipo 2, píxeles no de campo en zona de campo son mucho más numerosos que los errores de tipo 1, dado que muchos de estos píxeles no son errores sino elementos del juego, zona restringida o publicidad sobre el campo que obviamente presentan un color distinto al campo. Por ello no resulta trivial la comparación entre ambos tipos de errores a la hora de llegar a un criterio.

En esta versión se trató de mejorar la forma en la que se selecciona este candidato encontrando una forma de pesar ambos errores de la forma más optima posible. Para ello, se aplica un factor sobre el error de tipo 1 (píxeles de campo por encima de los bordes), que proporcione los mejores resultados posibles. El valor de dicho factor calculado experimentalmente se presenta en la tabla 3.6.

La gran diferencia entre estos valores es debida a que en el caso de la detección de borde

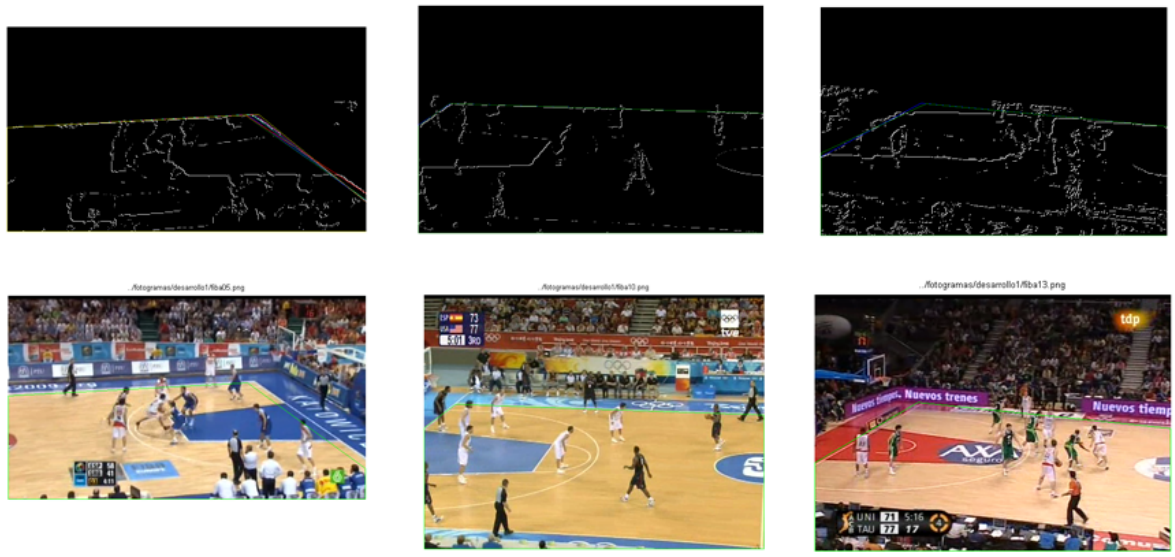

Figura 3.32: Ejemplo de la aplicación de la transformada de Hough a los bordes de tres fotogramas y de la nueva técnica de extracción de líneas para identificar el borde lateral.

	factor error superior
Borde superior	20
Borde lateral	200

Tabla 3.6: Factor de ajuste del error superior con respecto al error inferior

lateral el número de errores de tipo 1 (número de píxeles de campo fuera de la zona segmentada) puede ser mucho menor ya que la segmentación de borde superior deja únicamente una pequeña sección triangular de posibles errores. Por lo tanto, dado que unos pocos píxeles de campo fuera de la segmentación de campo en la detección del borde lateral son mucho más importantes que lo muchos píxeles no de campo que quedaran dentro de la misma (que en general serán debidos a los citados elementos del juego presentes sobre la cancha y no a una segmentación deficiente), es necesario darles un mayor peso a dichos errores de tipo 1. En la figura 3.33 se ve un ejemplo de uno de estos casos patológicos.

Cuando la máscara no incluya una sección tan grande de errores en la parte superior como en este caso, también se cumple que este peso funciona bien, debido a la forma en la que se comparan los bordes.

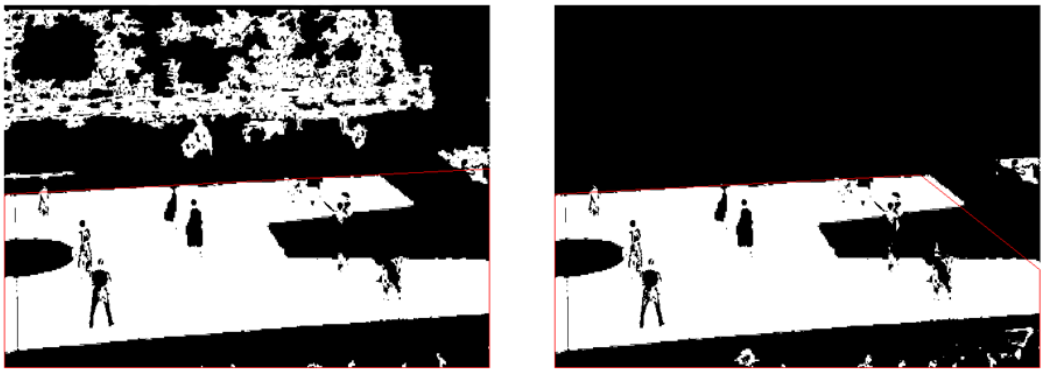

Figura 3.33: Ejemplo de las máscaras de las que se cuentan los errores para seleccionar un buen candidato. Se ve que en la imagen de la derecha (la usada para seleccionar el borde superior) existen muchos más errores en la parte superior que en la de la derecha.

En primer lugar ya que tenemos construidos nuestros polígonos candidatos podemos ordenarlos antes de aplicarles el criterio para que nos quedemos con el más apropiado. La ordenación es la siguiente:

- Borde Superior: Ordenamos de según la componente vertical (y) del primero punto. De tal forma que si en media la pendiente de los bordes es descendente o cero (probable borde izquierdo o medio) ordenamos del borde del primer punto más bajo al borde con el primer punto más alto, si la pendiente es ascendente (probable borde derecho) ordenamos de manera inversa. De esta forma al aplicar el criterio de selección nos quedaremos con el borde más alto según el sentido de las pendientes.

- Borde lateral: Ordenamos según la componente horizontal (x), de tal manera que cuando trabajemos con bordes laterales izquierdos los ordenemos de menor a mayor y cuando trabajemos con bordes laterales derechos los ordenemos de mayor a menor. Así, al aplicar el criterio de selección nos quedamos con el borde más interior que lo cumpla.

En el caso del borde superior sumamos todos los píxeles de error tanto superiores como inferiores y seguidamente los sumamos entre si con los errores superiores debidamente pesados. El criterio es el siguiente, por cada borde candidato nos quedamos con él, si su error (tal y como se calcula en la ecuación 3.12) es como mucho 1.05 veces el error del mejor candidato encontrado hasta entonces. Este error con el que comparamos es inicializado a ∞.

$$error_{borde} = errores_{pixeles_{campo}} + peso_{error_{nocampo}} \cdot errores_{pixeles_{nocampo}} \tag{3.12}$$

De esta forma nos quedamos con el borde más superior posible siempre y cuando su error sea menor (con un 5 % de toleracia) que el error del mejor de los otros bordes candidatos. Esto se hace así para que el sistema en casos dudosos tienda a seleccionar el borde más superior.

En el caso de los bordes laterales hay que tener en cuenta que ya tenemos una segmentación, la que contempla sólo la existencia de un borde superior (sin borde lateral visible), adicional a los posibles candidatos. Por ello, a pesar de que el criterio de la pendiente nos diga que es probable que el borde lateral sea visible, sólo elegiremos un candidato a borde lateral si este reduce el error que se obtendría si se usara la segmentación sin borde lateral. En cualquier caso, el umbral aplicado como criterio de pendiente (que detecta en que lado del campo nos encontramos) varía ligeramente del expuesto en 3.3.7, teniendo ahora el valor:

$$umbral_{lado} = 0{,}03 \tag{3.13}$$

Al igual que en la versión anterior si la pendiente es mayor que $umbral_{lado}$ buscaremos un borde lateral izquierdo, si es menor que $-umbral_{lado}$ buscaremos un borde lateral derecho y si está entre estos dos valores el siguiente bloque no hará nada, ya que la presencia de un borde lateral no es factible.

El error que contabilizamos de cada borde se construye de una forma similar al caso del borde superior pero simplemente tendremos en cuenta los errores que se producen en la pequeña área triangular que queda entre el anterior borde superior y el actual borde lateral (ver figura 3.34), ya que asumimos que el borde superior está correctamente detectado. Esta aproximación será estudiada con más profundidad más adelante, analizando su conveniencia.

Por lo tanto, el error se calcularía tal y como se describe en 3.14. En esta ecuación x_5 es la coordenada x del vértice del polígono donde se cortan ambos bordes (ver ecuación 3.7 y figura 3.25) es decir donde se pasa de un tipo de segmentación a la otra, en consecuencia el *ancho del área de cálculo error* serán el número de píxeles que tiene en sentido horizontal (de ancho) este área triangular en la que estamos detectando los bordes.

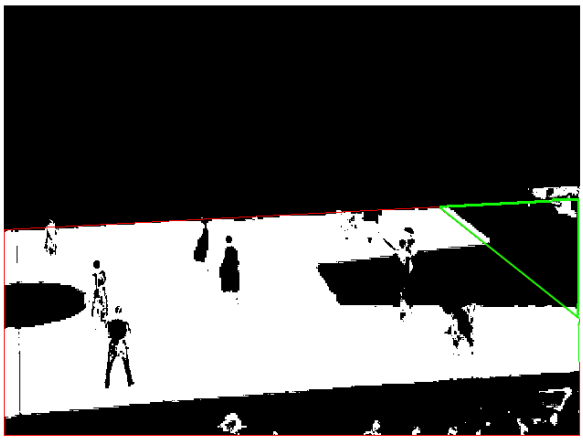

Figura 3.34: En la figura se puede ver el área que queda entre el borde superior y el candidato a borde lateral (marcada en verde). El ancho de esta zona se usará en el cálculo de errores de borde lateral.

$$error_{borde} = \frac{errores_{pixeles_{campo}} + peso_{error_{nocampo}} \cdot errores_{pixeles_{nocampo}}}{ancho\ del\ area\ calculo\ error}$$

$$ancho\ del\ area\ calculo\ error = \begin{cases} x_5 & \text{si borde lateral izquierdo} \\ x_{max} - x_5 & \text{si borde lateral derecho} \end{cases} \tag{3.14}$$

Se elige un candidato frente a la segmentación sin borde lateral si:

- Mejora al mejor de los bordes anteriores. Su error tal y como se describe en 3.14, es menor que el del mejor de los otros candidatos.
- El error inferior del candidato, $\frac{errores_{pixeles_{campo}}}{ancho\ del\ area\ calculo\ error}$, es menor que el error que se obtendría de utilizar simplemente la segmentación con borde superior, calculado como: $\frac{errores_{pixeles_{campo}}}{\frac{ancho\ de\ la\ imagen}{2}}$

El segundo criterio tiene en cuenta que el ancho del área en el que se calcula el error será como mucho la mitad del ancho de la imagen, por lo que en el peor de los casos el error tiene que se igual al que se obtiene con el borde superior dividido entre la mitad de la imagen. Este criterio es muy poco restrictivo para que sólo se aplique en casos en los que los que todos bordes laterales sean completamente erróneos, casos en los que se mantendrá la segmentación obtenida sin borde lateral.

Como se verá en el análisis de resultados de la sección siguiente esta segunda aproximación a la selección de candidatos mejora notablemente la versión anterior aunque presenta algunos errores que analizaremos en dicha sección.

3.4.5. Análisis de resultados

Si, al igual que se hizo en la sección anterior, aplicamos al sistema con las mejoras descritas en las secciones anteriores el banco de prueba de desarrollo y valoramos como se describió en la sección 3.2 la tasa de acierto en la detección del borde superior, la tasa de acierto en la detección del lado de campo en la que nos encontramos y la tasa de acierto en la detección del borde lateral obtenemos los siguientes resultados:

Resultados de la versión 0.2			
	Detección de borde superior	**Clasificación de lado de campo**	**Detección de borde lateral**
Test 1	80 %	88 %	84 %
Test 2	92 %	83,2 %	75,2 %

Tabla 3.7: Resultados del banco de pruebas de desarrollo al aplicarlo en la versión 0.2

En la figura 3.35 comparamos los resultados expuestos en la tabla 3.7 con los obtenidos en la versión anterior (tabla 3.4).

Como se observa claramente en la figura se ha observado una clara mejora en los resultados fruto de los cambios que se implementaron y que se han descrito en este capítulo. A continuación (en la figura 3.36) se puede ver una comparativa de fotogramas que daban mal resultado en la versión 0.1 y ahora con la versión 0.2 obtenemos segmentaciones correctas.

El primero de los casos se beneficia del refinamiento de la función que extrae el color del campo con la corrección de saturación descrita en la sección 3.4.1. En el segundo fotograma lo que resulta decisivo es la delimitación del rango de la transformada de Hough en el que buscamos los picos, así como la mejora en la función de detección de candidatos a frontera de campo (ambas técnicas descritas en 3.4.3). La tercera segmentación es obtenida de manera exitosa gracias principalmente a la nueva función de extracción de bordes (sección 3.4.2) y en la mejora del criterio de selección de candidatos (sección 3.4.4).

A pesar de la mejora experimentada existen algunos errores que persisten de la versión ante-

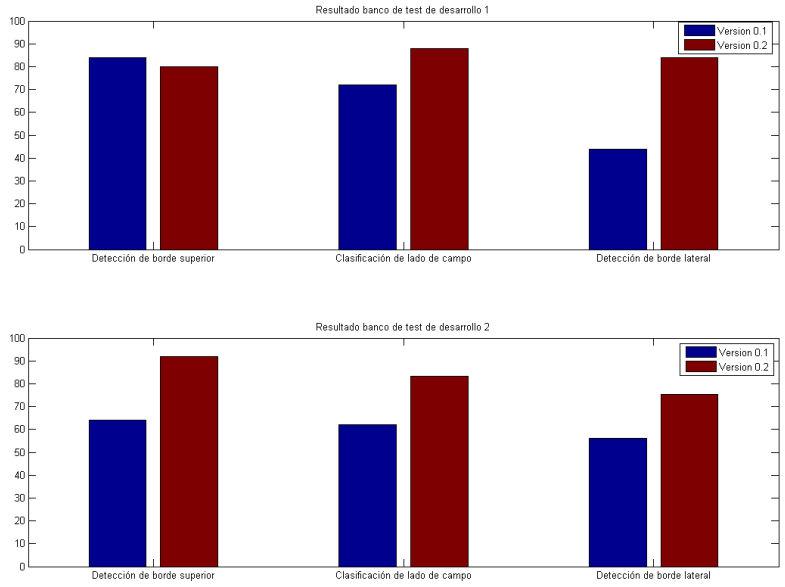

Figura 3.35: Comparación de resultados entre la versión 0.1 y 0.2.

rior. En la siguiente figura (3.37) se observan algunos de estos ejemplos.

En el primero de los casos vemos que se ha detectado correctamente el borde superior pero que al tener una pendiente suficientemente pronunciada parece factible que exista un borde lateral, este es detectado debido a la disposición de la zona restringida que presenta una recta muy clara. Parece claro que la segmentación sin borde lateral es notablemente mejor, ya que deja muy pocos errores por encima, y nuestro criterio no es suficientemente estricto con las segmentaciones laterales ya que sólo las rechaza si son mucho peores que la que sólo incluye borde superior. Por lo tanto, aunque en esta versión existe una clara mejora del sistema se puede plantear una mejora del criterio de selección (especialmente en la selección de borde lateral) que permita la comparación objetiva de errores. Dicha mejora se implementó en la versión final.

El segundo fotograma presenta un problema relacionado con el hecho de que tal y como se construyen las máscaras un falso borde superior es identificado como el mejor porque efectivamente deja menos errores. Si este borde además tienen una pendiente muy diferente a la del borde superior real provoca que el borde lateral se trate de detectar incorrectamente o ni siquiera se busque si la pendiente tiene un valor absoluto menor que el umbral (con es el caso). Estos casos son difíciles de solucionar, sobretodo en máscaras como éstas en las que el rellenado elimina gran parte del campo. La mejor forma de solucionarlas es teniendo en cuenta cierta información de

Figura 3.36: Ejemplo de fotogramas en los que se ha experimentado una mejora en la nueva versión. En la primera fila tenemos los resultados al aplicarles la versión 0.1, en la segunda los resultados al aplicar a los mismos fotogramas el sistema descrito en la versión 0.2 (en ambos casos la segmentación está representada con una línea verde).

otras segmentaciones anteriores, añadiendo un cierto procesado temporal. Además en este caso esta segmentación incorrecta se presenta sola en una secuencia sin fallos en el resto de fotogramas, por lo que el procesado temporal parece especialmente factible.

En el tercer caso tenemos una suma de todos los posibles problemas. Por un lado los bordes son sumamente rugosos, debido al poco contraste en H entre la zona de campo y la pintura en torno a él. En segundo lugar aparece el problema del caso 2, tenemos un borde similar al correcto que tiene una pendiente suficientemente diferente como para, al igual que el caso primero, trate de buscar el borde lateral. Este borde lo encuentra y lo da por válido al igual que en el fotograma de la primera columna.

Por lo tanto parece que como planteamiento para la siguiente versión se extraen la siguientes posibles mejoras:

- Añadir algún tipo de procesado temporal que corrija al vuelo aquellas segmentaciones malas que ocurran en secuencias correctas.

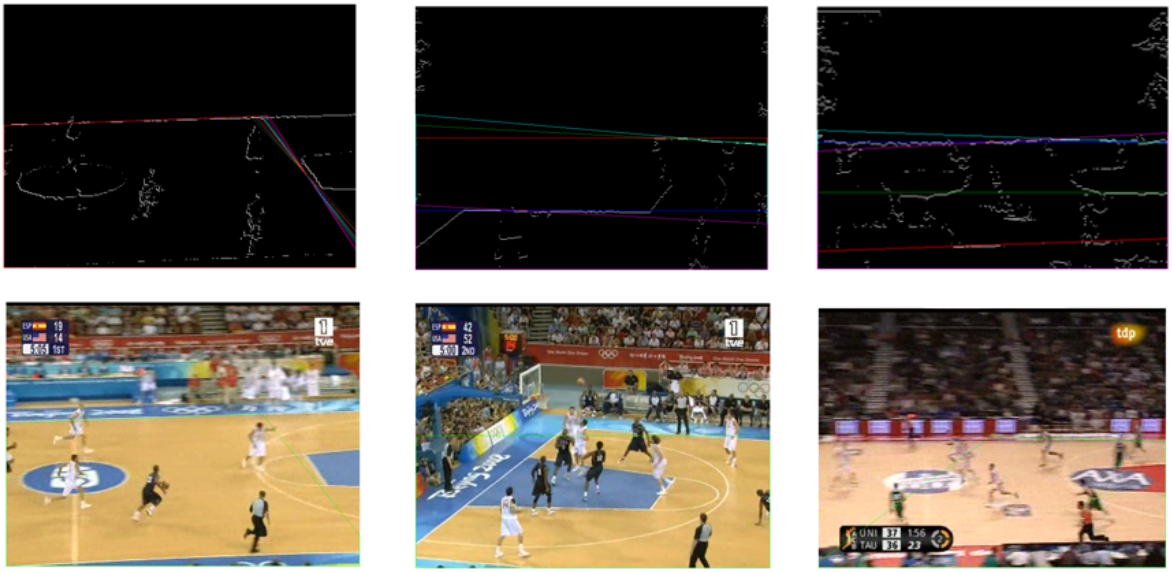

Figura 3.37: Algunos ejemplos de los errores que nos encontramos a aplicar el sistema en la versión 0.2 a los bancos de pruebas

- Mejorar el criterio de selección de forma que no valide un borde lateral que no mejore el borde superior y que además tenga en cuenta el procesado temporal anterior.

En cuanto al procesado temporal, en el test de desarrollo 2, que como se ha descrito anteriormente consiste en secuencias de 1 segundo de duración, se observa que la mayoría de los problemas de segmentación aparecen en ráfagas cortas de dos o tres fotogramas separadas por fotogramas bien clasificados. En algunos casos incluso las segmentaciones erróneas son fotogramas sueltos en secuencias que se comportan bien. En ambos casos si encontrásemos una forma de aprovechar la información extraída de segmentaciones anteriores que consideremos correctas se podrían corregir la mayor parte de estos fallos.

Este procesado temporal es la base de la siguiente mejora incorporada en la versión 0.3. La dificultad principal de este procesado es como detectar automáticamente que las segmentaciones anteriores son más correctas que la actual. Por ello, y dado que el criterio de selección de candidatos de esta versión tiene los problemas descritos, se modificará dicho criterio para obtener una medida válida de calidad de la segmentación que nos permita por un lado mejorar cada segmentación individualmente y por otro lado compararla con las obtenidas en secuencia para

actuar en consecuencia.

3.5. Versión Final. Versión 0.3

En esta sección se describirá la última versión del sistema segmentador representado por el diagrama de bloques 3.2. Con respecto a la versión anterior y basándonos en los resultados expuestos en 3.4.5, se ha añadido una forma de tener en cuenta segmentaciones de fotogramas anteriores para corregir la segmentación actual (sección 3.5.1) y se ha refinado el criterio de selección de bordes candidatos para evitar problemas como los descritos en la versión anterior (ver 3.4.5) y permitir una comparación objetiva de segmentaciones de fotogramas sucesivos.

Por último en el apéndice C está descrita la arquitectura *software* de la aplicación, en esta versión, con una explicación de cada una de las funciones implementadas y de la relación entre ellas.

3.5.1. Descripción e implementación del mecanismo de procesado temporal

Una vez implementadas las mejoras de la versión anterior y viendo que existían una serie de casos problemáticos se decidió por utilizar la valiosa información temporal de las secuencias de vídeo. La idea en la que nos basamos es la siguiente: En una secuencia de vídeo de baloncesto (secuencia principal tal y como describimos en 3.1) la zona de campo en un fotograma será muy similar o igual a la del fotograma anterior.

El cambio de posición del campo de un fotograma a otro de la misma secuencia viene dado por la velocidad a la que la cámara se mueve, típicamente siguiendo el ritmo del juego. Así en ataques rápidos en los que los jugadores se desplacen velozmente por el campo, la cámara se moverá deprisa y por lo tanto la zona de campo cambiará más entre un fotograma y el siguiente mientras que en un ataque estático la zona de campo será la misma en una serie de fotogramas consecutivos. Aún así, dado que entre dos fotogramas median 40 milisegundos de tiempo (la tasa de fotogramas es 25 fotogramas por segundo y no es factible que sea menor que eso en ninguna aplicación real), la variación en el peor de los casos será pequeña. Esto se puede ver claramente en la figura 3.38 donde se muestran los dos casos extremos (extraídos de los resultados de la versión 0.2): un ataque estático y un contraataque rápido.

Por lo tanto si conocemos la segmentación de campo de un fotograma y estamos seguros de que es correcta y obtenemos una posible segmentación en el fotograma siguiente que con alta probabilidad es incorrecta, podemos utilizar la primera segmentación para calcular la de ese fotograma.

Figura 3.38: Dos ejemplos de transiciones dentro de una misma secuencia. La primera de ellas en un ataque estático en la que la camara apenas se mueve, la segunda un ataque rápido donde el cambio es mayor.

Existen diversas aproximaciones a este problema de diversa complejidad. Como hemos hecho anteriormente primaremos soluciones simples frente a otras más complejas, para mantener el nivel de eficiencia del sistema. Por ello nuestra aproximación tendrá únicamente en cuenta la segmentación obtenida en el fotograma anterior que, si comparada con la segmentación obtenida del fotograma actual la mejora, substituirá a esta última. Una correcta segmentación anterior será una buena segmentación para el fotograma actual en tanto en cuanto la posición de la zona de campo no varíe mucho entre dos fotogramas. Como hemos visto en la figura 3.38, esta sencilla técnica nos ha de permitir corregir a partir de un fotograma correctamente segmentado una ráfaga de aproximadamente 4-8 fotogramas erróneos en el caso peor (caso en el que la cámara se mueva más velozmente). Obviamente podríamos haber desarrollado una técnica más compleja a esta (descrita en el diagrama 3.39) que utilizara segmentaciones de diversos fotogramas sucesivos para mejorar la segmentación actual mediante por ejemplo técnicas de estimación de la posición de la misma, pero dado que la variación entre dos fotogramas es tan pequeña y viendo que el rendimiento del sistema en la versión anterior deja ráfagas de error de unos pocos fotogramas optamos por esta aproximación con un sólo fotograma de memoria.

El mayor problema que nos encontramos es como medir objetivamente lo buena que es una segmentación y compararla con la segmentación siguiente. En este sentido es muy importante evitar utilizar erróneamente una mala segmentación para corregir segmentaciones actuales por lo que se le dará al sistema una cierta inercia a considerar en los casos dudosos la segmentación

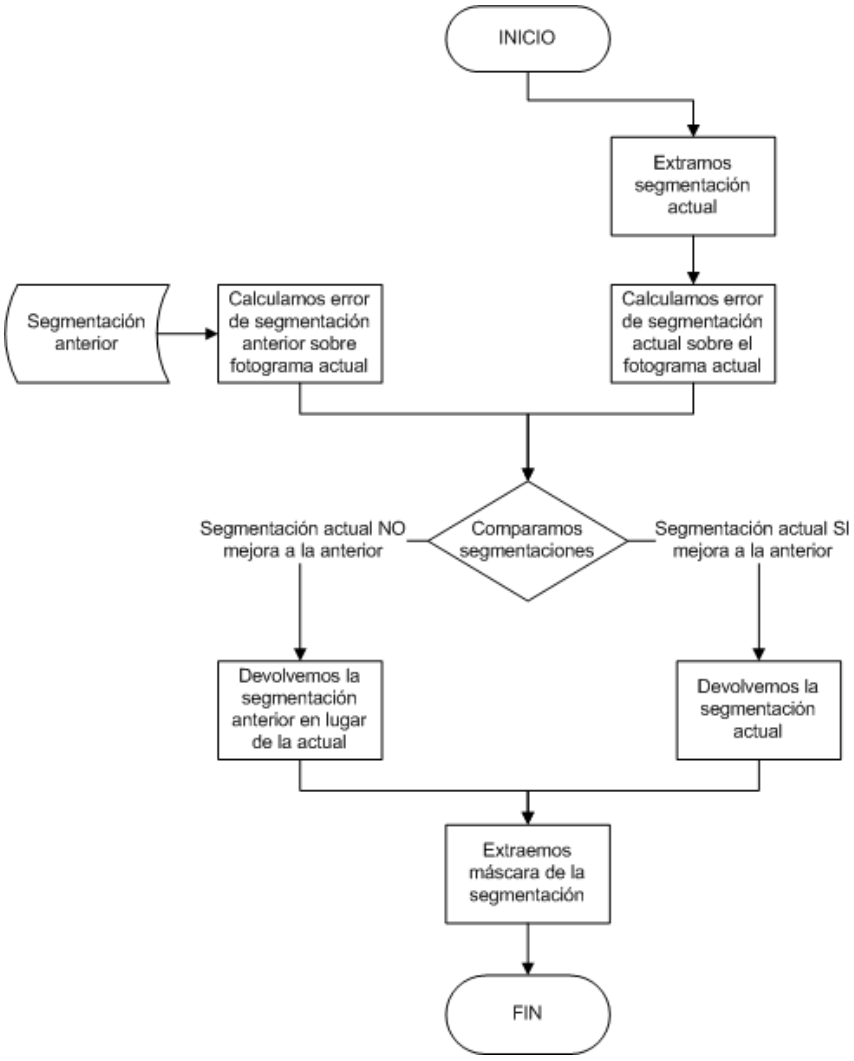

Figura 3.39: Diagrama de flujo del proceso de elección entre la segmentación anterior y la actual.

actual como mejor que la anterior para no arrastrar un error durante muchos fotogramas. Hay que considerar que para contar errores disponemos de una máscara de campo que a veces representa deficientemente la realidad del campo por ello hay que tratar que el criterio usado minimice este problema. Este criterio de selección está descrito en la sección siguiente.

3.5.2. Criterio final de selección de candidatos contemplando el procesado temporal

Como se ha visto en secciones anteriores el criterio de selección de los candidatos es crítico. Dado que no somos capaces de extraer una máscara más precisa a partir del color del campo,

debido a tonalidades similares entre diversas zonas, es importante que al menos sepamos identificar cuál es el borde que más se parece al real a partir de aquellos que tenemos como candidatos. Además debemos ser capaces de identificar en que casos nuestra mejor segmentación no es buena para sustituirla, en caso de que esta sea mejor, por la segmentación anterior.

La mayor dificultad que presenta esta fase es, que debido a la existencia de elementos dentro de la zona del campo que en la máscara no aparecen como de campo (jugadores, carteles, zonas de reflejos muy fuertes,etcétera), se contabilizan muchos errores en la parte interior de la posible zona de campo que no serán deberían ser errores. Esto provoca que en ciertos casos patológicos no nos quedemos con el mejor borde posible. Si este error se da al detectar el borde superior, dicho error probablemente provoque que no seamos capaces de detectar el borde lateral (por ejemplo en casos en los que el borde superior detectado tiene una pendiente muy diferente al real y por lo tanto buscamos el borde lateral en una zona incorrecta). En estos casos es donde cobra mayor sentido utilizar la segmentación anterior ya que, si esta es correcta, probablemente presentará un error mucho menor que la actual (donde el borde lateral está calculado a partir de una información de borde superior errónea) y por lo tanto podremos utilizarla en lugar de la mal calculada segmentación actual.

El principal problema que presenta esta aproximación de sustitución de una segmentación por la anterior es, cuando se dan algunas condiciones perjudiciales, el sustituir una segmentación correcta por una errónea porque el error contabilizado por esta última es menor. En general cuando suceda esto la diferencia entre ambos errores será pequeña por lo que podemos esperar que el error cometido por elegir un borde u otro en estos casos no sea muy grande en estos casos. Pero en todo caso como el sistema tiene una alta tasa de acierto al segmentar el campo, a tenor de las pruebas realizadas anteriormente (ver 3.4.5), al sistema se le impondrá una inercia a elegir la segmentación actual frente a la anterior.

Asimismo, cuanto más diferente sea una segmentación de la anterior, más probable es que haya que sustituirla por la misma, ya que existen más probabilidades de que sea errónea. Por ello esta inercia de la que hablábamos antes a elegir la segmentación actual en lugar de la anterior será función de la diferencia entre dos segmentaciones tal y como se explicará más adelante.

A partir de lo explicado en los párrafos anteriores se fijaron los siguientes criterios de selección. En el caso del borde superior mantenemos la forma de comparar descrita en la sección 3.4.4. Utilizando la misma tolerancia (5 %) y el mismo peso para los errores superiores (20). Dicho criterio parece funcionar bien para la mayoria de los casos y sólo falla en los casos en los que la

máscara de color es muy mala, ya sea porque el borde es demasiado rugoso para obtener una respuesta (casos en los que típicamente obtenemos sólo un borde en la parte inferior de la zona restringuida que suele tener una respuesta más clara), o porque existen demasiados píxeles negros debido al rellenado de la máscara en la zona de campo y estos llevan a seleccionar un borde que segmenta de tal forma que deja todos los posibles fuera.

Es en el caso del borde lateral donde se concentran los cambios en el criterio. En primer lugar hay que contemplar el borde de la anterior segmentación. Para ello separamos la selección de candidatos en dos pasos:

1. Seleccionamos al mejor de los candidatos de la segmentación actual sin tener en cuenta la segmentación anterior según el criterio expuesto más adelante en la ecuación 3.15.

2. Comparamos el mejor de los candidatos actuales con la segmentación actual, comparando sus errores y teniendo en cuenta la diferencia que existe entre ambas segmentaciones.

El primer punto se basa en lo desarrollado en la versión anterior con la salvedad de que ahora el error no está escalado por el tamaño de la zona de cálculo (ver 3.4.4). En este caso asumimos que la segmentación superior es correcta y por lo tanto simplemente comparamos los distintos errores que presentan los bordes laterales calculados, es decir la cantidad de píxeles no de campo por debajo del borde lateral y la cantidad de píxeles de campo por encima del mismo. Dado que ya no dividimos por este ancho de zona, ni contamos los errores de nuevo del borde superior, el peso que se le aplica a los errores de encima del borde lateral ha de ser más bajo, por lo que lo fijamos a 30. De esta forma obtenemos un error que podemos comparar más fácilmente con el del borde anterior. Elegiremos el borde que nos de un menor error de entre todos los candidatos.

$$\begin{gathered} error_{borde} = errores_{pixeles_{campo}} + peso_{error_{nocampo}} \cdot errores_{pixeles_{nocampo}} \\ peso_{error_{nocampo}} = 30 \end{gathered} \tag{3.15}$$

Como se ve ya no se compara con los errores del borde superior como se hacia antes. Se observó que generalmente, aunque la pendiente del borde superior indicara que estábamos en uno de los laterales, si no aparecía ese borde en la imagen la transformada de Hough no devolvía respuesta. En el caso de que una segmentación no tenga borde lateral y erróneamente encontremos uno este será corregido por el procesado temporal, ya que la segmentación anterior (donde ese borde no aparece) presentará, por lo general, mucho menos error por debajo del borde.

La comparación con el borde anterior se hace en los siguientes términos. Se calcula el error de ambos bordes como la suma del error del lado derecho y la suma del error del lado izquierdo, la separación entre ambos lados será el punto 5 del polígono de segmentación (si es una segmentación sin borde lateral sólo habrá lado derecho). Los errores se calcularán del modo descrito en la ecuación 3.15 y por lo tanto no será necesario recalcular el error de uno de los lados para la segmentación actual. Finalmente se seleccionará el borde según el criterio siguiente:

$$\begin{aligned} error_{actual} &= error_{izquierdo_{actual}} + error_{izquierdo_{actual}} \\ error_{anterior} &= error_{izquierdo_{anterior}} + error_{derecho_{anterior}} \\ area_{diff} &= \frac{\sum_{pixeles} |mascara_{actual} - mascara_anterior|}{area_{total}} \end{aligned} \tag{3.16}$$

$$Criterio : error_{actual} < \frac{1{,}3 \cdot error_{anterior}}{1 + 5 \cdot area_{diff}}$$

Donde $area_{diff}$ es la suma de todos los píxeles diferentes entre ambas segmentaciones dividida por el número total de píxeles. Esto se calcula de manera parecida a como se hacia en la versión 0.1 3.3.7, calculando el área de la imagen binaria que obtenemos al restar ambas máscaras. Como se dijo en dicha sección, en Matlab podemos obtener una máscara a partir del polígono de bordes con la función *poly2mask* y a partir de ella calcular el número de píxeles igual a 1 con *bwarea*. Los otros dos valores (1.3 y 5) se ajustaron empíricamente.

El valor de $area_{diff}$ es típicamente mucho menor que 1. En aquellos casos donde hay un cambio notable entre las segmentaciones tiende a ser superior a 0.05. Por ello al multiplicar por un factor 5, eliminamos la inercia a elegir como mejor borde al borde actual fijada por el valor (1.3). En todo caso pequeños cambios en estos valores llevan a cambiar errores o aciertos en las decisiones, por lo que su valor fue ajustado para maximizar el éxito con el banco de pruebas utilizado, un mayor número de pruebas permitiría sin duda un ajuste más fino de estos parámetros.

3.5.3. Análisis de Resultados

Una vez realizadas las modificaciones descritas se aplicó, al igual que en las secciones 3.3.9 y 3.4.5, el banco de pruebas de desarrollo al sistema completo obteniendo los resultados que se pueden ver en la tabla 3.8.

En la figura 3.40 comparamos los resultados expuestos en la tabla 3.8 con los obtenidos en

Resultados de la versión final			
	Detección de borde superior	**Clasificación de lado de campo**	**Detección de borde lateral**
Test 1	80 %	84 %	84 %
Test 2	93 %	100 %	76 %

Tabla 3.8: Resultados del banco de pruebas de desarrollo al aplicarlo en la versión 0.3.

la versiones anteriores (tablas 3.4 y 3.7).

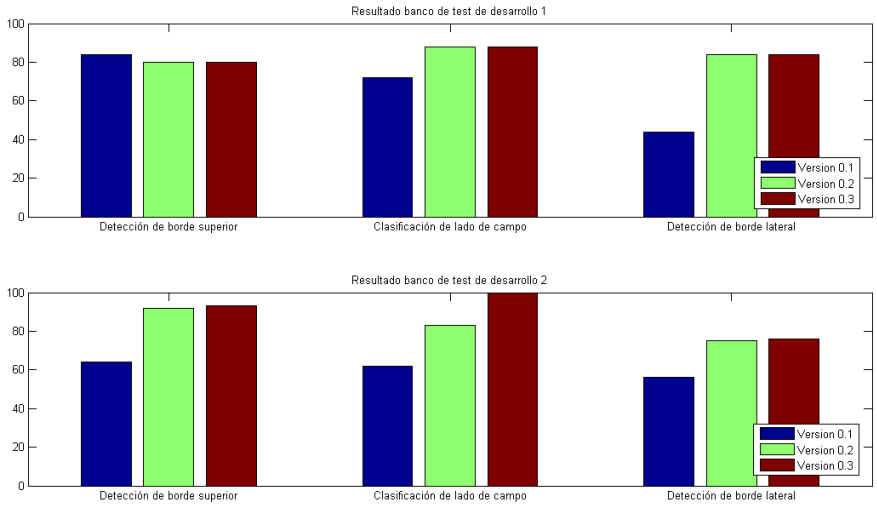

Figura 3.40: Comparación de resultados entre la versión 0.1, 0.2 y 0.3.

Como se ve en el caso del primer test al ser fotogramas sueltos el procesado temporal no aplica y se obtienen los mismos resultados que en la versión anterior. Lo que nos interesa más es el test 2, donde al estar los fotogramas en secuencia vemos el efecto de la nueva funcionalidad añadida: los resultados mejoran bastante en la clasificación del fotograma. Es cierto que la detección del borde lateral y superior no mejoran mucho pero globalmente esta aproximación es bastante acertada.

En la siguiente figura (figura 3.41) se pueden ver algunos de los casos en los que la aplicación de este procesado temporal mejora el sistema y algunos de los casos en los errores que no es capaz de solucionar. En ella se representan la segmentación elegida con el mismo color verde del resto de figuras y mediante una línea punteada roja la segmentación que se obtendría si no se aplicara este procesado temporal.

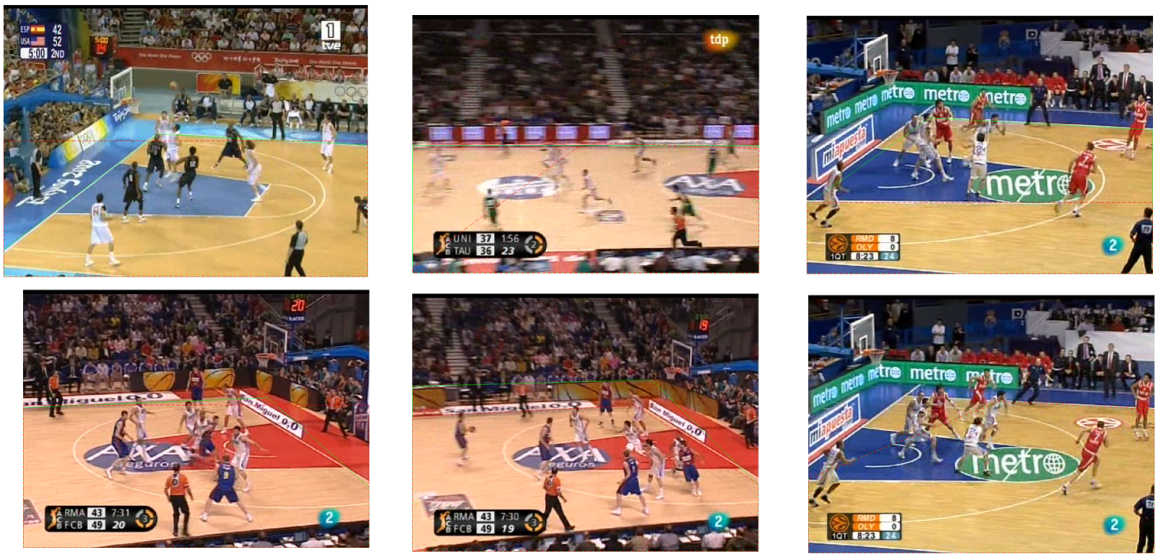

Figura 3.41: Ejemplo de algunos de los errores que corrige esta nueva versión (fila superior) y de algunos que no es capaz de corregir (fila inferior).

En cuanto a los errores que es capaz de solucionar, por ejemplo tenemos el primer caso (esquina superior izquierda) que es un error aislado en una secuencia que por otra parte se comporta bien, en la segmentación actual no se encuentra ningún borde debido a la situación de los jugadores, luces, etcétera y por lo tanto se elige la segmentación anterior. En el segundo y tercer caso tenemos dos secuencias con una serie de errores en ráfaga (en el primer caso la transición es bastante rápida y por lo tanto la posición del campo varía más deprisa que en otros casos) de los cuales somos capaces de corregir la mayoría, aunque en la segmentación anterior no sea exacta la consideramos suficientemente buena.

En cuanto a los errores que no solucionamos cabe destacar que la mayor parte de los problemas aparecen en el vídeo 4 del banco de desarrollo (ver tabla 3.1), debido a las características del campo: color del borde exterior, de los carteles de publicidad, condiciones de luminosidad, etcétera (primer y segundo caso de la fila inferior). Este vídeo concentra casi el 70 % de los errores de segmentación de borde lateral (siendo sólo el 20 % de los fotogramas analizados). Esta situación se analizará con más profundidad en la sección siguiente (3.6) en la que se realiza a esta versión final alguna prueba más de eficiciencia y robustez, pero en todo caso hay que destacar el buen resultado que da el sistema con el banco de pruebas que se ha usado en su diseño y que existen

casos que resultan de especial problema para él, al obtener una máscara de color que representa peor al campo:

- Campos con el borde de color rojizo, cuyo valor de H está muy próximo al del parquet del campo.
- Condiciones en las que la luminosidad provoca reflejos en torno a los bordes del mismo, haciendo más rugosos los bordes al extraer la máscara.
- Situaciones en las que la posición de los jugadores, zona restringida y carteles de publicidad durante varios fotogramas (típicamente más de 5) hace que el rellenado de la máscara elimine gran parte del campo. Si esto pasa en ráfagas cortas se puede solucionar mediante la utilización de la segmentación anterior. Este problema será obviamente mayor cuanto más grandes se vean los jugadores (más cercano sea el plano cuya distancia cambia entre diferentes pabellones).

De estos casos se puede concluir qué secuencias y vídeos presentarán una mejor o peor respuesta al sistema y esta información será la que usemos para medir la robustez del mismo al aplicar el sistema a secuencias típicamente problemáticas en la siguiente sección.

3.6. Pruebas de evaluación y eficiencia

Como se ha podido ver en las secciones anteriores el sistema funciona bastante bien cuando lo probamos con las secuencias y fotogramas con los que lo hemos desarrollado. Estas pruebas no nos sirven para medir la calidad del sistema por eso mismo, por lo que tenemos que utilizar un banco de pruebas diferente que llamaremos banco de evaluación. En la sección 3.2 se describe en que consiste este banco de pruebas y de que vídeos se extraen las secuencias, vídeos diferentes a los utilizados en el otro banco. Con este banco de pruebas realizaremos un análisis de la calidad y robustez del sistema y pruebas de eficiencia en las que se medirán los tiempos de cómputo.

3.6.1. Pruebas de calidad y robustez

Para comprobar si el sistema es robusto lo aplicaremos a un banco de pruebas de otros partidos diferentes a los usados en el desarrollo del mismo. Realizaremos las mismas pruebas que en las secciones 3.3.9, 3.4.5 y 3.5.3, midiendo la tasa de acierto en la detección del borde superior, la clasificación de lado de campo y la detección de borde superior, pero sólo del equivalente al test 2, es decir aquel que prueba el sistema completo al utilizar secuencias de 1 segundo de duración. En la siguiente tabla (tabla 3.9) y gráfica (figura 3.42) podemos ver esos resultados y su comparación con los obtenidos anteriormente con el test 2 del banco de desarrollo.

Resultados evaluación de la versión final			
	Detección de borde superior	**Clasificación de lado de campo**	**Detección de borde lateral**
Test Evaluación	74 %	84 %	64 %

Tabla 3.9: Resultados del banco de pruebas de evaluación.

Se puede ver que el resultado con este banco de pruebas es algo peor que con el banco de desarrollo. Si observamos los resultados obtenidos en cada una de las secuencias (ver figura 3.43) vemos que la mayoría de los problemas se concentran en un secuencia en concreto para las dos pruebas, especialmente en el caso del banco de evaluación donde una secuencia no puede ser segmentada en absoluto. Esto es algo que llevamos viendo desde el principio del desarrollo, por sus características existen ciertos vídeos que se comportan mucho peor que otros.

Esto es un problema para la robustez del sistema ya que no parece ser válido para todo tipo

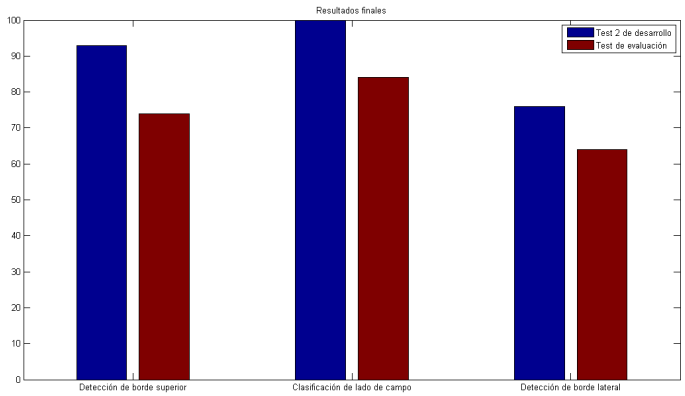

Figura 3.42: Comparativa de resultados obtenidos con las pruebas del bando de desarrollo y el banco de evaluación.

de partidos, más concretamente presenta problema en aquellos vídeos en los que el contraste en tonalidad entre el campo y el borde del mismo es poco claro. En general es un problema si se dan además las condiciones de iluminación y saturación inadecuadas. En la figura 3.44 mostramos algunos fotogramas del tipo de vídeos que peores resultados dan de entre las pruebas. Es visible a simple vista la similitud del matiz de ambas zonas del campo especialmente en el caso del segundo fotograma, que además es el que peor resultado proporciona como se ve en las gráficas de la figura 3.43.

En este sentido se puede plantear para futuras líneas de investigación el análisis de estos partidos problemáticos, y otros, para mejorar el sistema o estudiar otras aproximaciones alternativas que no presenten estos problemas. En todo caso si analizamos críticamente las medidas obtenidas vemos que si exceptuamos ese tipo de partidos que presentan tan mal rendimiento, en el resto de los casos logramos una segmentación buena o muy buena (como se dijo en la sección 3.2 las pruebas realizadas tienen una componente subjetiva y decidimos considerar que una segmentación era buena en base a lo bien que se ajustara al borde real, siendo bastante estrictos en este sentido) en cerca del 80 % de los casos. Pensamos que es un resultados suficiente para plantearse la posibilidad de profundizar más en este método en futuras investigaciones fuera del ámbito de este proyecto.

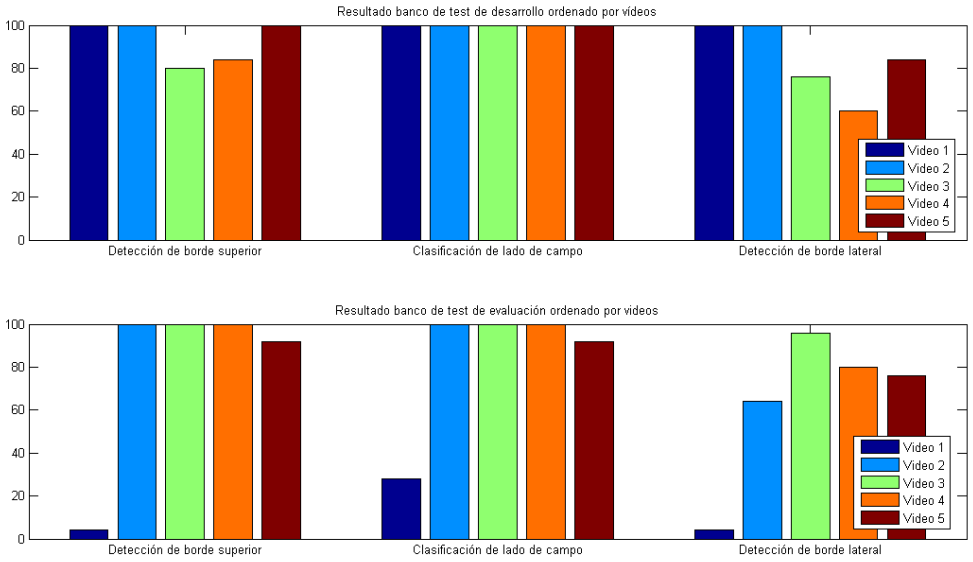

Figura 3.43: Resultados del banco de pruebas de desarrollo y evaluación separado por secuencias de vídeo. Es importante destacar que los vídeos de uno y otro banco son diferentes.

3.6.2. Pruebas de eficiencia

Para realizar un análisis de la eficiencia del sistema y su viabilidad para integrarlo en un sistema de detección de eventos en tiempo real se realizaron las medidas de tiempo de cómputo del sistema implementado en Matlab en los dos ordenadores personales descritos en la sección 1.4.

La medida que se realizó del tiempo de cómputo se hizo con el test 2 del banco de prueba de desarrollo y el banco de evaluación, que disponen de 125 fotogramas cada uno de diferente resolución. Cada banco completo se ejecutó 5 veces para disponer de un número más representativo de valores de tiempo, ya que este varía en función del resto de aplicaciones que se estén ejecutando en el ordenador y del sistema operativo.

En esta medida sólo se contó el tiempo desde que el fotograma está disponible (incluida su lectura de fichero) hasta que el sistema termina. Por lo tanto no se contabilizó el tiempo empleado en extraer el fotograma del vídeo que como se ha dicho en 3.3.1 es un proceso bastante lento debido a las funciones utilizadas y podría ser implementado de una forma muy eficiente en un sistema real en *Hardware*. Tampoco se muestra ninguna figura durante la ejecución de estas

Figura 3.44: Estos son fotogramas de las secuencias que peor resultado dan en los tests. El primer fotograma es de la secuencia 4 del banco de desarrollo, los otros dos son de la secuencia 1 y 2 respectivamente del banco de evaluación.

pruebas dado que la generación y representación de las mismas es también bastante lenta. Los resultados obtenidos se muestran en la tabla 3.10.

Tiempos de cómputo medio de cada secuencia de vídeo		
	Ejecución en PC1	**Ejecución en PC2**
Secuencia de desarrollo 1	0.2552 s	0.3001 s
Secuencia de desarrollo 2	0.3579 s	0.4751 s
Secuencia de desarrollo 3	0.2510 s	0.3749 s
Secuencia de desarrollo 4	0.2934 s	0.3583 s
Secuencia de desarrollo 5	0.2333 s	0.3139 s
Secuencia de evaluación 1	0.6273 s	0.7198 s
Secuencia de evaluación 2	0.8840 s	1.1763 s
Secuencia de evaluación 3	0.5796 s	0.9487 s
Secuencia de evaluación 4	0.6768 s	1.5448 s
Secuencia de evaluación 5	0.5495 s	1.6813 s
Tiempo medio total	0.4708 s	0.7893 s

Tabla 3.10: Tiempos medios de cómputo de los distintos vídeos probados, tanto de desarrollo como de evaluación, en los dos PC usados en las pruebas.

En esta prueba denominamos PC1 al ordenador usado unicamente en estas pruebas de eva-

luación, el segundo equipo descrito en la sección 1.4, un equipo de sobremesa *Medion* con un procesador *Intel Core i5 (CPU 750 @ 2.67 Ghz)* con 6GB de memoria DDR3 y *Windows 7 Home Premium*. EL PC2 es el principal PC usado en el desarrollo y es un ordenador portátil *Acer Aspire* con un procesador *Intel Core 2 Duo T7200 (2.0 GHz,667 Mhz FSB, 4 MB L2 cache)* con 1GB de memoria DDR2 y *Windows XP Media Center Edition* como sistema operativo.

Como se ve el tiempo medio de cómputo es 0.4708 s y 0.7893 s para cada uno de los ordenadores. Existe una diferencia considerable entre ambos debido a la diferencia de potencia entre ambos equipos, lo que nos lleva a pensar que si este mismo sistema funcionara en un equipo más potente o en el que no tuviera que compartir el procesador con otra serie de procesos, este podría mejorarse notablemente.

En todo caso con estos tiempos de cómputo podríamos procesar un fotograma cada aproximadamente 400 milisegundos. Si conserváramos la tasa de 25 fps, esto significaría que podríamos procesar unos 2 o 3 fotogramas de cada segundo para mantener dicha tasa. Como se ha visto anteriormente en el caso peor observamos un cambio en la posición del campo cada 5 o 6 fotogramas por lo que para conservar las prestaciones del sistema al menos deberíamos procesar uno de cada 3 fotogramas y poder seguir sacando ventaja del procesado temporal. Viendo los resultados obtenidos, para lograr esto solamente habría que mejorar un poco estos tiempos. Por lo que una implementación en un equipo más potente y dedicado únicamente a esta función, viendo el notable cambio entre uno y otro equipo, probablemente lograra funcionar en tiempo real.

En el caso de que esto no fuera así siempre podríamos implementar en código nativo alguna de las funciones que más tiempo consumen (y que ya no estén precompiladas en Matlab). Si hacemos un estudio de las funciones que consumen más tiempo (a través de la herramienta *profiler* de Matlab) observamos que algunas candidatas a ello son:

- *upperborder*.
- *imfill*.
- *sideborder*.
- *rgb2hsv*.

Entre ellas consumen más de la mitad de tiempo de cómputo del sistema. De estas funciones *upperborder* y*sideborder* están implementadas por nosotros y su mayor carga está en las múltiples

operaciones necesarias para el cálculo de errores y selección de candidatos (la transformada de Hough es bastante rápida en comparación con esto, alrededor de un 10 % del tiempo de *upperborder*). Ambas podrían ser implementadas en C/C++ e integrarlas con el código Matlab mediante el interfaz MEX y de esta forma hacerlas más eficientes. Lo mismo ocurre en el caso de la función *rgb2hsv*, aunque en este caso es una función de *Image Processing Toolbox* de Matlab también está implementada puramente en lenguaje Matlab en lugar de una función MEX con lo que potencialmente se reduciría su tiempo de cómputo. En el caso de *imfill* y otras funciones como *imdilate*, también bastante lentas, la mayoría de su implementación ya está hecha en lenguaje nativo por lo que su rendimiento sólo se podría mejorar mediante el uso de un entorno más potente. Estas dos funciones realizan operaciones morfológicas sobre la imagen binaria que constituye la máscara y dichas operaciones son potencialmente complejas computacionalmente hablando.

Con todo esto podemos concluir que si bien los resultados no nos permiten afirmar con seguridad la viabilidad de este sistema para su uso en detección de eventos en tiempo real, si nos hacen ser optimistas en cuanto a sus posibilidades.

3.6.3. Otras pruebas

Además de las pruebas realizadas quisimos hacer alguna prueba más para analizar la viabilidad de extender, con ligeros cambios, el trabajo a otros ámbitos; así como analizar que parámetros del sistema resultan más críticos a la hora de ser utilizados con vídeos de otros deportes. Estas pruebas no pretenden ser intensivas sino darnos una idea de las posibilidades del sistema.

Como se dijo en la sección 3.2 el ámbito del proyecto son los partidos de baloncesto FIBA y no los de NBA debido a las diferencias existentes entre los campos de las dos corrientes. Esto se decidió al principio del desarrollo al ver las diferencias que se obtenían entre los resultados de uno y otro tipo. Por ello queríamos, ahora que tenemos una versión final implementada, probar el sistema con algunos fotogramas de partidos NBA. En la figura 3.45 se muestra el resultado de aplicar el sistema desarrollado (sin realizar ningún cambio en sus parámetros) a tres fotogramas de partidos NBA.

Como se aprecia hay un fotograma (el segundo) que es segmentado correctamente mientras que en los otros dos existen fallos. En el primer caso el borde superior es correctamente segmentado pero el borde lateral aparece detrás del correcto debido a que el borde de alrededor del

Figura 3.45: Resultado del sistema al aplicarlo sobre 3 fotogramas de NBA para los que no está diseñado

campo no está pintado de otro color. Este problema no parece que pueda ser solucionado con el ajuste de ninguno de nuestros parámetros por lo que habría que idear alguna técnica que lo solventara. En el tercer caso el borde superior es detectado incorrectamente, por lo que el lado del campo se clasifica mal y no se detecta el borde superior. El problema que aquí aparece también lo encontramos en partidos FIBA y es debido a que la máscara no representa bien el campo tras su rellenado (debido a la posición de los jugadores se rellena parte del campo) y no obtenemos el borde más correcto de entre los candidatos. Esto se soluciona casi siempre con el procesado temporal ya que esa segmentación aunque sea elegida como el mejor candidato a borde superior no constituye un borde mejor que uno correcto detectado anteriormente.

Por lo tanto en muchos partidos NBA parece que el sistema funcionaría correctamente, para otros debido a las diferencias en el campo (en NBA no es obligatorio que el borde del campo o la zona estén pintados) habría que desarrollar una técnica diferente quizás apoyada por la detección de algún elemento más del campo (zona restringida, mástil de la canasta, etcétera).

Si somos algo más ambiciosos y pretendemos extenderlo a otros deportes en los que el campo también presente una tonalidad uniforme podemos pensar por ejemplo en fútbol. En la figura se ve el resultado al aplicarlo sobre un fotograma de este deporte.

Para obtener este resultado exitoso tuvimos que modificar ciertos parámetros del sistema en las siguientes partes:

- **detección del color de campo**. Hubo que modificar el rango de H (R_h1) para ponerlo en torno al color verde. Se ajustó a los valores $R_h1 = [0{,}2\ ,\ 0{,}3]$.

- **detección de borde superior**. Hubo que eliminar el mínimo ρ en el que se busca en la transformada de Hough para permitir detectar bordes más altos. Como el borde está más

Figura 3.46: Resultado de aplicar el sistema (con una serie de parámetros modificados) a un fotograma de fútbol.

alto, el numero de errores superiores es notablemente más alto y para detectar el borde correcto fue necesario aumentar notablemente el peso de esos errores a alrededor de 10000.

- **detección de borde lateral**. Hubo que ampliar el rango de θ en el que se busca el borde en la transformada de Hough, ya que dada la situación de la cámara los bordes pueden estar más inclinados. Simplemente se añadieron 10° a cada extremo del rango, fijándolo a [10° , 80°].

De estas escasas pruebas no se puede concluir que el sistema sea válido para partidos de baloncesto NBA o otros deportes como el fútbol, pero si que podemos establecer que con unos pocos ajustes parece adaptarse bastante bien a estos otros casos por lo que podría ser interesante analizar sus posibilidades fuera del ámbito de este proyecto.

Capítulo 4

Conclusiones y Líneas futuras

4.1. Conclusiones del proyecto

De todo el estudio, diseño, implementación y pruebas expuestos anteriormente se pueden extraer una serie de conclusiones sobre el resultado obtenido y el grado de cumplimiento de los objetivos presentados en la sección 1.3.

- Del estudio del estado del arte previo concluimos que no existe en el campo del procesado de vídeos de baloncesto gran desarrollo en cuanto a la detección de eventos en tiempo real. La mayoría de la investigación existente se centra en crear marcos de trabajo para la detección multimodal (apoyándose también en el procesado del audio de la retransmisión, además del vídeo) y en la clasificación del tipo de plano. En cuanto a la segmentación de campo no existen tampoco trabajos centrados específicamente en baloncesto y lo que existe en otros deportes se basa en herramientas difícilmente implementables automáticamente y en tiempo real. Eso dota por si mismo de importancia a este proyecto como primer paso para la implementación de un sistema de detección de eventos en vídeos de baloncesto.

- Si nos centramos en el sistema diseñado, este constituye una interesante aplicación de diversas técnicas de procesado de imagen como son la transformación entre espacios de color, la extracción de máscaras e imágenes de bordes o el trabajo con la transformada de Hough a un caso real. Observando problemas típicos de la segmentación de imágenes por regiones no adscritos a este ámbito en concreto.

- Los resultados obtenidos muestran la complejidad del problema enfrentado pero nos per-

miten ser optimistas en cuanto a su desarrollo futuro. Los resultados son muy buenos en clasificación del lado del campo y en detección del borde superior, aunque no lo son tanto a la hora de generar la segmentación completa. La mayoría de los problemas se presentan en un subconjunto de vídeos que han sido identificados y descritos en el desarrollo, este hecho es considerado positivo a pesar de comprometer la robustez del sistema ya que probablemente permitirá en un desarrollo futuro, centrarnos en ellos y mejorar el sistema de manera importante ya que si eliminamos ese subconjunto se obtienen buenos resultados.

- En cuanto a la eficiencia del sistema, al no haber sido implementado en un entorno real, no podemos asegurar su viabilidad en la integración en un sistema en tiempo real. Pero las herramientas de las que disponemos y las pruebas realizadas también nos llevan al optimismo en cuanto a sus posibilidades.

- Por último el sistema parece resolver, con unos mínimos cambios, de manera bastante buena problemas de segmentación similares al planteado en el proyecto pero fuera del ámbito del mismo, como son partidos de baloncesto NBA o de deportes de hierba. Esto nos lleva a pensar que las técnicas aquí aplicadas son suficientemente generales y aplicables a otros problemas similares con un ajuste de un número limitado de parámetros del mismo, alguno de los cuales han sido identificados a lo largo del desarrollo.

Por lo tanto como conclusión final se puede afirmar que, aunque los resultados obtenidos no cumplen al 100 % los objetivos planteados (especialmente en cuestión de robustez), si que son bastante positivos si consideramos que es una primera aproximación a un problema complejo. Además los problemas existentes han sido identificados y cuantificados. Por lo que consideramos al sistema una buena base para futuros desarrollos en segmentación de campo en vídeos deportivos.

4.2. Líneas futuras

En este proyecto se ha presentado una primera aproximación a la segmentación eficiente de campos de baloncesto. Aunque nos basamos en investigaciones previas, la mayor parte del sistema fue desarrollado de manera práctica combinando diferentes técnicas por ello deja numerosas líneas de investigación abiertas que podrían ser desarrolladas a partir del mismo.

Estas líneas de investigación pueden ser clasificadas en tres grandes grupos. En primer lugar la mejora del sistema implementado para obtener unos mejores resultados en su mismo ámbito de aplicación. En segundo lugar la integración de este sistema en uno más amplio de detección de eventos. Por último, la aplicación del sistema aquí desarrollado (con los cambios necesarios) a otros ámbitos.

4.2.1. Mejoras en el sistema

A pesar de que los resultados obtenidos se consideran positivos, quedan una serie de problemas y funcionalidades abiertas para nuevos desarrollos. A continuación exponemos las que consideramos más importantes:

- Llevar a cabo un análisis estadístico de los posibles casos que nos podemos encontrar en los vídeos de retransmisiones deportivas (en términos de posición del campo, distribuciones de color, distancia de plano, etcétera) permitiría ajustar los parámetros del sistema de manera más adecuada. Para ello sería necesario utilizar un banco de pruebas más amplio que el utilizado en este proyecto y diversas herramientas de estimación.

- Desarrollar una métrica objetiva que permita medir de la manera más automática posible la calidad de la segmentación, para sustituir la métrica (bien o mal segmentado) utilizada aquí, útil aunque subjetiva.

- Ampliar las funcionalidades del sistema para que identifique más puntos de interés del campo como pueden ser la base de la canasta, la zona restringida, etcétera; que pueden enriquecer la información disponible del juego y permitirnos calibrar la posición de la cámara. Esta calibración de la cámara abre un gran abanico de posibilidades tanto en el desarrollo de nuevas funcionalidades de extracción de características de "alto nivel" como en mejora de la detección del campo.

- Reimplementar aquellas funciones del sistema que presenten peor eficiencia de manera que se pueda garantizar su funcionamiento en tiempo real.
- Mejorar el algoritmo de procesado temporal para lograr (sin aumentar en exceso la complejidad computacional del sistema) un aprovechamiento eficaz de la gran correlación existente entre fotogramas consecutivos.

4.2.2. Integración en un sistema completo de detección de eventos

Como se definió al principio, este desarrollo está pensado para funcionar en un sistema de detección en tiempo real. Por ello sería necesario integrarlo con otros desarrollos que permitan la extracción de características del vídeo necesarias para la detección de eventos.

En el diagrama 1.2 se plantea un marco de trabajo para este tipo de sistema. El subsistema desarrollado en este proyecto encuadraría en el bloque de *procesado de plano principal lateral*. En este mismo bloque podríamos encuadrar otros subsistemas como la citada calibración de cámara, detección de jugadores, identificación de equipos, detección de balón, etcétera.

Además sería necesario desarrollar un método de clasificación de planos que distinguiera que tipo de secuencia es cada una de las presentadas en este sentido hay trabajo realizado en los artículos de Ekin y Murat Tekalp [10] y [9].

4.2.3. Extensión de las técnicas estudiadas a otros contextos

Como se vio en 3.6.3 la mayor parte de las técnicas aquí desarrolladas son suficientemente generales como para ser aplicadas a otros problemas similares de segmentación de campo en vídeos deportivos. A lo largo de la memoria se han tratado de remarcar aquellos parámetros cruciales y como afectaban a la funcionalidad del mismo, dichos parámetros son los candidatos a ser ajustados para permitir la extensión del sistema a otros problemas.

Por el contrario también se vio que existen una serie de características de los campos de baloncesto FIBA (como son el gran contraste en matiz entre la cancha y el borde de la misma) que son importantes para el buen funcionamiento del mismo. Por lo tanto, probablemente sea necesario desarrollar alguna técnica nueva que permita utilizarlo para resolver problemas que no presenten estas características. Aún así consideramos el trabajo realizado como un buen punto de partida a la hora de enfrentar este tipo de problemas.

APÉNDICES

APÉNDICE A

Presupuesto del proyecto

En este apéndice se presentan justificados los costes globales de la realización de este Proyecto Fin de Carrera. Tales costes, imputables a gastos de personal y de material, se pueden deducir de las Tablas A.2 y A.3.

Fases del proyecto		
Fase 1	*Documentación*	105 *horas*
Fase 2	*Desarrollo del software*	399 *horas*
Fase 3	*Redacción de la memoria del proyecto*	150 *horas*
Total		654 *horas*

Tabla A.1: *Fases del Proyecto*

En la Tabla A.1 se muestran las fases del proyecto y el tiempo aproximado para cada una de ellas. Así pues, se desprende que el tiempo total dedicado por el proyectando ha sido de 654 horas, de las cuales aproximadamente unas 54 han sido compartidas con el tutor del proyecto. Estas horas están contabilizadas y cuantificado su coste en la tabla A.2. En ella la figura Hombre-mes equivale a $131,25$ horas. Ascienden a un total de $15{,}187,77$ €.

En la Tabla A.3 se recogen los costes en equipos informáticos, calculando la amortización correspondiente a los meses de trabajo utilizados. Ascienden, pues, a un total de 180 €.

Coste directo por personal				
Apellidos y nombre	**Categoría**	**Dedicación (hombres-mes)**	**Coste (€/hombre-mes)**	**Coste (€)**
Jesús Cid-Sueiro	Ingeniero Senior	0,4	4,289,54	1,715,82
Jesús Fernández Bes	Ingeniero	5	2,694,39	13,471,95
Hombres-mes		5,4	**Total**	15,187,77

Tabla A.2: *Costes directos de personal*

Coste directo por equipos					
Descripción	**Coste (€)**	**% Uso dedicado proyecto**	**Dedicación (meses)**	**Periodo de depreciación**	**Coste imputable**
PC1	900,00	100	1	60	15,00
PC2	1,100,00	100	9	60	165,00
	2000	**Total**			180,00

Tabla A.3: *Costes directos de equipos informáticos*

A partir de estos datos, el presupuesto total es el mostrado en la tabla A.4.

Coste Total	
Concepto	**Importe**
Costes personal	15,187,77 €
Costes directos amortización de equipos	180 €
Costes indirectos (20 %)	3,074 €
TOTAL	18,441 €

Tabla A.4: *Presupuesto total*

El presupuesto total de este proyecto asciende a la cantidad de 18,441 euros.

Alcorcón a 5 de Agosto de 2010

El ingeniero proyectista

Fdo. Jesús Fernández Bes

APÉNDICE B

El espacio de color HSV

En el presente apéndice se presenta una breve introducción al espacio de color HSV, si se desea profundizar más en los espacios de color, especialmente en el modelo RGB con el que a continuación se presupone cierta familiaridad, se recomienda acudir a *Digital Image Processing with Matlab* de Gonzalez [12] o a *Computer graphics and geometric modeling: implementation and algorithms* de Agoston [1].

El propósito de los espacios de color es facilitar la especificación del color según algún tipo de estándar generalmente aceptado. En esencia un modelo de color no es más que un sistema de coordenadas 3-D y un subsistema de él donde cada color es representado por un punto. Los modelos de color suelen estar orientados hacia el *Hardware* y a como representar el color en monitores o impresoras (caso de los modelos RGB, CMY o YIQ) o hacia aplicaciones que tienen como objetivo manipular el color. En este segundo grupo entran los modelos HSI (*Hue, Saturation, Intensity*) y HSV(*Hue,Saturation,Value*).

El modelo HSV es un modelo mucho más cercano a la forma que tienen los humanos de percibir el color que el modelo RGB, de hecho los conceptos que representan sus 3 componentes son aproximadamente a lo que en pintura se denomina como matiz, tono y luminosidad (aunque dicha correspondencia es más clara con el modelo HSI/HSL muy similar a este). Por esta razón es un modelo adecuado para implementar algoritmos que traten de imitar sistema de visión humano.

Desde el punto de vista formal se puede definir a partir del modelo RGB mediante una transformación no lineal: Es la proyección del cubo RGB vista a lo largo de su eje gris desde el blanco hacia el negro. La figura B.1 compuesta de varias otras extraídas de la Wikipedia [20][19] y de

los libros [12] y de [1] puede resultar esclarecedora:

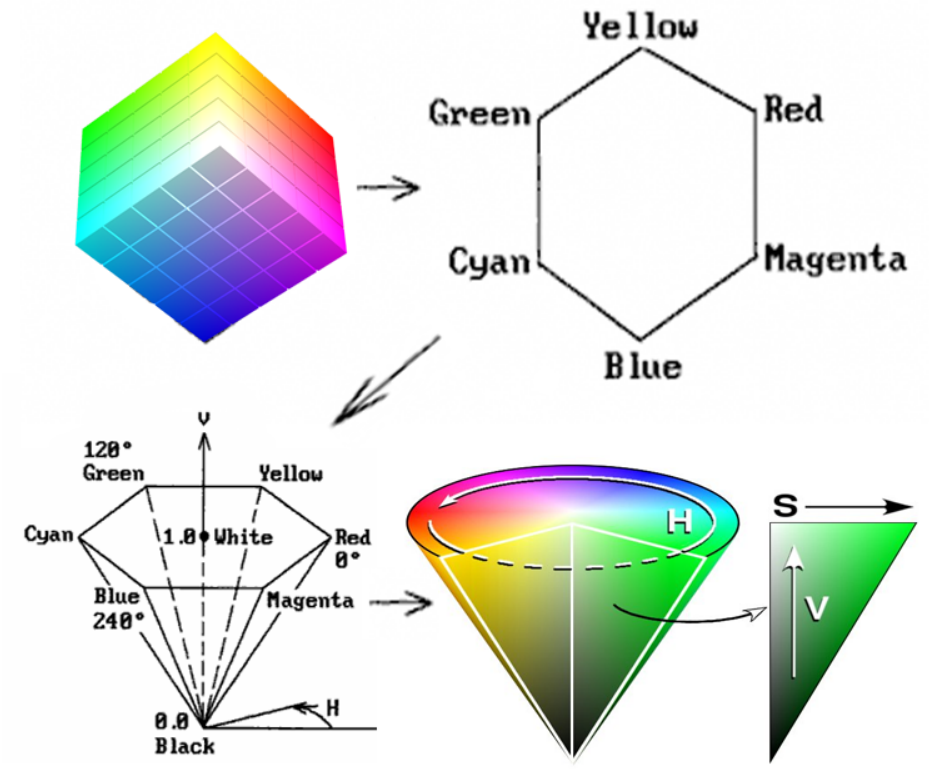

Figura B.1: Paso de RGB a HSV. Extraída a partir de [20], [19], [12] y [1].

Como se ve la *Hue* es una componente angular que parte desde el rojo (0^o) con el cian en 180^o. La componente *Saturation* es una componente radial que va desde el color más aclarado al color puro y la componente *Value* va desde el negro al blanco en el eje del cono y por lo tanto de tonalidades más oscurecidas a más puras en el resto.

En Matlab se dispone de sendas funciones *rgb2hsv* y *hsv2rgb* para llevar a cabo estas proyecciones entre los dos espacios de color. Las ecuaciones B.1 y B.2, extraídas del código fuente de estas funciones, presentan las transformaciones que llevan a cabo para transformar de un espacio a otro. Hay que tener en cuenta que en esta implementación de Matlab, las tres componentes HSV tienen valores comprendidos entre 0 y 1, es decir en H los 0^o (rojo) corresponderán a 0, el cian a 0.5 y el 1 a rojo de nuevo. En S y V el valor 1 significa máxima saturación y brillo respectivamente.

- Ecuaciones de la transformación del espacio RGB al HSV:

$$
\begin{aligned}
V &= max(R,G,B) \\
C &= \begin{cases} 1 & \text{si } max(R,G,B) = min(R,G,B) \\ max(R,G,B) - min(R,G,B) & \text{en otro caso} \end{cases} \\
H' &= \begin{cases} \frac{1}{6}\frac{G-B}{C} & \text{si } R = max(R,G,B) \\ \frac{2}{6}\frac{R-B}{C} & \text{si } G = max(R,G,B) \\ \frac{4}{6}\frac{R-G}{C} & \text{si } B = max(R,G,B) \end{cases} \\
H &= \begin{cases} H'+1 & H < 0 \\ H' & \text{en otro caso} \end{cases} \\
S &= \begin{cases} \frac{C}{V} & \text{si } max(R,G,B) - min(R,G,B) \neq 0 \\ 0 & \text{si } max(R,G,B) - min(R,G,B) = 0 \end{cases}
\end{aligned}
\tag{B.1}
$$

- Ecuaciones de la transformación del espacio HSV al RGB:

$$
\begin{aligned}
C &= V \cdot S \\
H_i &= 6 \cdot H \\
f &= h_i - \lfloor H_i \rfloor \\
p &= V - C(1-f) \cdot H \\
t &= V - C \\
n &= V - C \cdot f \\
(R,G,B) &= \begin{cases} (V,p,t) & \text{si } 0 <= H_i < 1 \\ (n,V,t) & \text{si } 1 <= H_i < 2 \\ (t,V,p) & \text{si } 2 <= H_i < 3 \\ (t,n,V) & \text{si } 3 <= H_i < 4 \\ (p,t,V) & \text{si } 5 <= H_i < 5 \\ (V,t,n) & \text{si } 5 <= H_i < 6 \end{cases} \\
(R,G,B) &= \frac{(R,G,B)}{MAX(R,G,B)}
\end{aligned}
\tag{B.2}
$$

Por último en la figura B.2 se presenta una representación extraída del sistema implementado en este proyecto en el que se aprecian las tres componentes HSV de un fotograma (siendo S=V=1, saturación y brillo máximos, para la representación de la componente H).

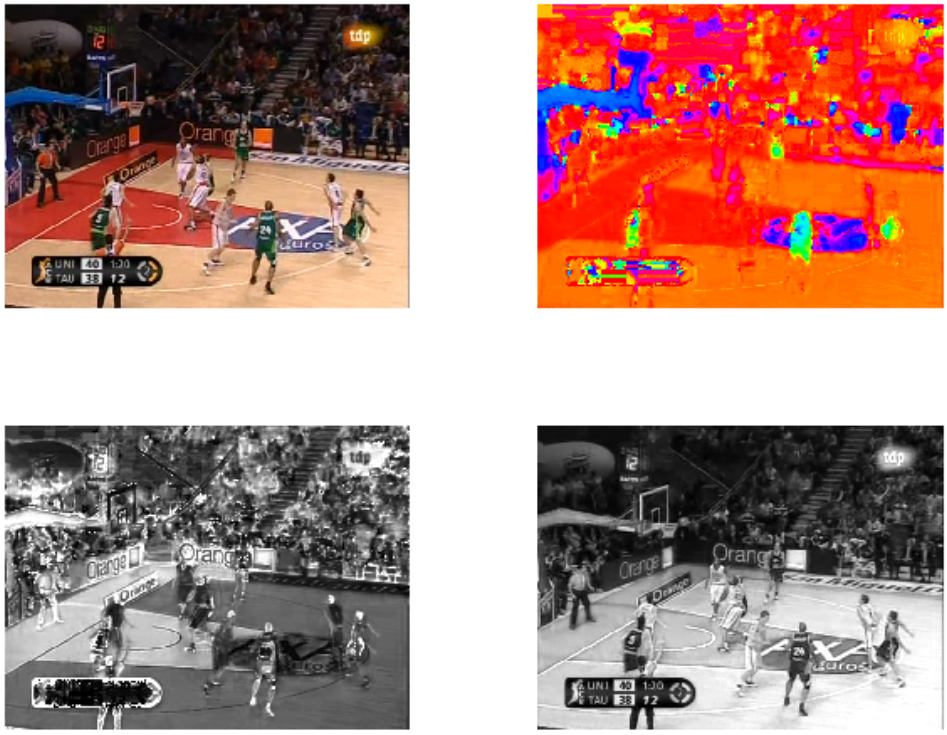

Figura B.2: Componentes HSV de un fotograma (De izquierda a derecha. Arriba: fotograma completo y componente H (S=V=1). Abajo: componente S y componente V.

APÉNDICE C

Arquitectura software de la aplicación

Como se ha comentado a lo largo de la presente memoria la aplicación esta implementada para la herramienta matemática Matlab. En el diagrama de la figura C.1 se muestra la arquitectura de funciones de la versión final de la aplicación (versión 0.3).

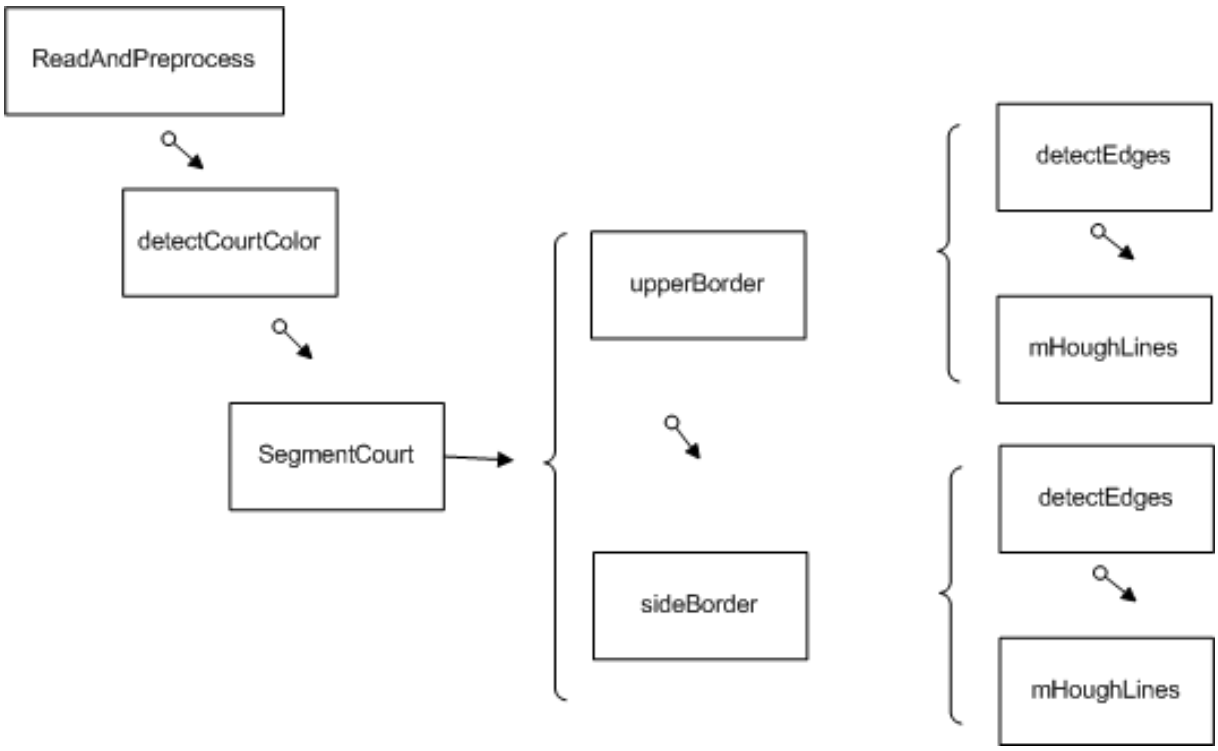

Figura C.1: Arquitectura de la aplicación. Sólo se representan las funciones implementadas en el presente proyecto.

A continuación se expone una breve descripción de las funciones implementadas.

- `[ I_rgb ] = readAndPreprocess( path )`

  Esta función lee un fotograma a partir de un fichero de imagen identificado por su *path* y devuelve una imagen en formato RGB preprocesada en forma de una matriz de dimensiones alto en píxeles X ancho en píxeles X 3, con elementos entre 0 y 1 representando el valor de cada una de las tres componentes RGB para cada píxel.

- `[ mask ] = detectCourtColor( I_rgb , debug )`

  Esta función encuentra el zona del color de campo dentro de una imagen RGB. Devuelve la máscara indicando esa zona. La máscara se representa como una matriz binaria de las dimensiones de la imagen. El parámetro *debug* indica si vale 1 que deseamos figuras e información de depuración.

- `[ borderCourt , maskCourt , current_border ]`
  `= segmentCourt( maskColor , old_border , debug )`

  Esta es la función principal que segmenta el campo de baloncesto a partir de una máscara de color. Además recibe el borde del fotograma anterior (*old_ border*) para mejorar la segmentación.El parámetro *debug* indica si vale 1 que deseamos figuras e información de depuración. Devuelve la frontera de la segmentación como dos vectores (x,y) de una componente por cada punto del polígono segmentador y su máscara asociada, además del borde que saldría si no se tuviera en cuenta *old_ border*.

- `[ up_border , mask_out , m ] = upperborder( mask_in , debug )`

  Esta función intenta obtener el mejor borde superior a partir de la máscara de color debidamente preparada (operación de rellenado adecuada). El parámetro *debug* indica si vale 1 que deseamos figuras e información de depuración. Devuelve la frontera de la segmentación sólo con borde superior, su mascara asociada y la pendiente de ese borde.

- `[ border_out , mask_out , current_border ]`
  `= sideborder( mask_in , border_in , m0 , old_border, debug )`

  Esta función intenta obtener la mejor segmentación (incluyendo el borde lateral si es necesario) a partir de la máscara de color debidamente preparada y el borde superior (*border_ in*).

El parámetro *m0* indica la pendiente del borde superior obtenida por *upperborder*. El parámetro *old_ border* es la frontera de la segmentación del fotograma anterior (si no está disponible su valor debe ser []. El parámetro *debug* indica si vale 1 que deseamos figuras e información de depuración. Devuelve la frontera de la segmentación *border_ out*, su mascara asociada *mask_ out* y la frontera *current_ border* que se obtendría si no se tuviera en cuenta *old_ border*, que puede ser igual a *border_ out* si la segmentación actual mejorara la que se obtendría con *old_ border*.

- `[ imageEdges ] = detectEdges( I_mask , type )`

  Esta función extrae los bordes de la máscara binaria. Se le tiene que indicar en el parámetro *type* que bordes queremos destacar: *'L'* para el borde izquierdo, *'R'* para el borde derecho y *'M'* para el borde superior. Develve una imagen de bordes en forma de una matriz binaria de dimensiones iguales a la de la máscara donde los bordes valen 1 y el resto de elementos (píxeles) valen 0.

- `[ borders , m ]`
  `= mHoughLines( P, rho , theta , width , height , type , varargin )`

  Función que sustituye a *houghlines* y que devuelve a partir de picos detectados en la transformada de Hough, polígonos candidatos a frontera de campo. Sus parámetros son esos picos *P*, los rangos de *rho* y *theta* de la transformada de Hough. El ancho *width* y alto *height* de la imagen en píxeles. El parámetro *type* es igual al de *detectEdges* y varargin{1} tiene que ser la frontera devuelta por *upperborder* si *type* es *'L'* o *'R'*. Devuelve los candidatos a bordes de la imagen en forma de dos matrices (x,y) en la que cada vector fila representa las componentes de una posible segmentación. Además devuelve un vector con las pendientes de todas las segmentaciones para posteriormente ordenar las segmentaciones en función de las mismas.

Además de estas funciones implementadas a lo largo del desarrollo del proyecto se utilizaron las siguientes funciones del *Image Processing Toolbox* en una o varias partes del programa:

- *im2double*
- *imshow*
- *imhist*
- *imfill*
- *imdilate*
- *strel*
- *hough*
- *houghpeaks*
- *imadjust*
- *mat2gray*
- *poly2mask*
- *bwarea*

Si se desea una descripción sobre las mismas recomendamos acudir a la documentación de Matlab.

Bibliografía

[1] M.K. Agoston. *Computer graphics and geometric modeling: implementation and algorithms*. Springer London, 2005.

[2] M. Barnard and J.M. Odobez. Robust playfield segmentation using MAP adaptation. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 3, 2004.

[3] M.H. Chang, M.C. Tien, and J.L. Wu. WOW: wild-open warning for broadcast basketball video based on player trajectory. In *Proceedings of the seventeen ACM international conference on Multimedia*, pages 821–824. ACM, 2009.

[4] Federacíon Española de Baloncesto. Reglas oficiales de baloncesto 2008. Aprobadas por el Comité Central de FIBA, Beijing (República Popular de China), Abril 2008.

[5] A.P. Dempster, N.M. Laird, D.B. Rubin, et al. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[6] R.O. Duda and P.E. Hart. Use of the Hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972.

[7] A. Ekin and A.M. Tekalp. Automatic soccer video analysis and summarization, August 1 2003. US Patent App. 10/632,110.

[8] A. Ekin and A.M. Tekalp. Robust dominant color region detection and color-based applications for sports video. In *International Conference on Image Processing*, volume 1, pages 21–4, 2003.

[9] A. Ekin and AM Tekalp. Shot type classification by dominant color for sports video segmentation and summarization. In *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03)*, volume 3, 2003.

[10] A. Ekin and M. Tekalp. Generic play-break event detection for summarization and hierarchical sports video analysis. In *Multimedia and Expo, 2003. ICME'03. Proceedings. 2003 International Conference on*, volume 1, 2003.

[11] D. Farin, S. Krabbe, HN Peter, and W. Effelsberg. Robust camera calibration for sport videos using court models. In *Proceedings of SPIE*, volume 5307, page 80, 2004.

[12] R.C. Gonzalez, R.E. Woods, and S.L. Eddins. *Digital image processing using MATLAB*. Prentice Hall Upper Saddle River, NJ, 2004.

[13] S. Jiang, Q. Ye, W. Gao, and T. Huang. A new method to segment playfield and its applications in match analysis in sports video. In *Proceedings of the 12th Annual ACM international Conference on Multimedia*, page 295. ACM, 2004.

[14] K. KIM, J. CHOI, N. KIM, and P. KIM. Extracting semantic information from basketball video based on audio-visual features. *Lecture notes in computer science*, pages 278–288, 2002.

[15] S. Liu, M. Xu, H. Yi, L. Chia, and D. Rajan. Multimodal semantic analysis and annotation for basketball video. *EURASIP journal on applied signal processing*, 2:32135, 2006.

[16] Y. Liu, S. Jiang, Q. Ye, W. Gao, and Q. Huang. Playfield detection using adaptive GMM and its application. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages 421–424, 2005.

[17] S. Nepal, U. Srinivasan, and G. Reynolds. Automatic detection of'Goal'segments in basketball videos. In *Proceedings of the ninth ACM international conference on Multimedia*, page 269. ACM, 2001.

[18] L. Wang, B. Zeng, S. Lin, G. Xu, and H.Y. Shum. Automatic extraction of semantic colors in sports video. In *IEEE International Conference on Acoustics, Speech, and Signal Processing, 2004. Proceedings.(ICASSP'04)*, volume 3, 2004.

[19] Wikipedia. Hsl and hsv — wikipedia, the free encyclopedia, 2010. [Online; accessed 20-May-2010].

[20] Wikipedia. Modelo de color hsv — wikipedia, la enciclopedia libre, 2010. [Internet; descargado 20-mayo-2010].

[21] M. Xu, L.Y. Duan, CS Xu, M. Kankanhalli, and Q. Tian. Event detection in basketball video using multiple modalities. In *Information, Communications and Signal Processing, 2003 and the Fourth Pacific Rim Conference on Multimedia. Proceedings of the 2003 Joint Conference of the Fourth International Conference on*, volume 3, pages 15–18. Citeseer, 2003.

[22] D. Zhong and S.F. Chang. Real-time view recognition and event detection for sports video. *Journal of Visual Communication and Image Representation*, 15(3):330–347, 2004.

[23] M. Zuliani. RANSAC for Dummies, 2008. http://vision.ece.ucsb.edu/ zuliani/Research/RANSAC/docs/RANSAC4Dummies.pdf.